SHARP

パソコンテレビX1
パーソナルコンピュータ

形名 CZ-800C

BASIC MANUAL

上手に使って上手に節電

目　　　次

第3章 関数・システム変数

付録

コマンド・ステートメント・関数

アルファベット順索引

ご　注　意

このマニュアルは、パーソナルコンピュータCZ-800Cの同梱BASICに基づいて作成されています。

(1)パーソナルコンピュータ CZ-800Cでは、システムソフトウェアを全てファイル形態のソフトウェアパック（カセットテープ）によってサポートされます。
各システムソフトウェアおよび本書の内容は、改良のため予告なく変更することがありますので、ファイルバージョンナンバーには、特にご注意されるよう、お願いいたします。

(2)パーソナルコンピュータ CZ-800Cのシステムソフトウェアならびに本書の内容を当社に無断で複製することは禁止します。

(3)本機は非常に複雑な機能および組合せを有する製品であり、出荷に際してマニュアルを含め充分なチェックをして万全を期しておりますが、万一ご使用中ご不審な点、お気付きのことがありましたら、もよりのシャープサービス・お客様ご相談窓口までご連絡ください。
なお、運用した結果、生じる影響については、責任を負いかねますので、あらかじめご了承ください。

SHARP HuBASICの特長

SHARP HuBASICは、以下の特長をもっています。

1．CZ-800Cのもつハードウェア機能を十分に活用できます。BASIC言語で操作可能なものとしては、以下のものがあげられます。

- ●プログラマブル・ファンクションキー
- ●専用カセット
- ●タイマー・カレンダーつきクロック
- ●セントロニクス・インターフェース機器
- ●テレビジョンコントロール
- ●グラフィック・プライオリティ・ロジック
- ●ユーザー定義キャラクタジェネレータ
- ●プログラマブル・サウンドジェネレータ

2．48Kバイトのグラフィック専用RAMにより各種のグラフィック処理に対応できます。
（1）640 ×200 ドット高分解能グラフィック
フルカラー8色1画面、または単色（8色中1色選択）3画面
（2）320 ×200 ドットマルチ画面
フルカラー8色2画面、または単色（8色中1色選択）6画面

3．WINDOW命令によって論理座標を設定することにより、簡単なプログラムで、多彩な表示を実現できます。

4．テキスト用VRAMとグラフィック用VRAMを別々にもっているため、テキスト画面とグラフィック画面の合成を容易に行えます。

5．パレットの概念によって、色の変更を瞬時に行うことができます。

6．特殊なハードウェアとそれらをサポートする強力なソフトウェアにより、グラフィック画面同士や、グラフィックとテキスト画面との間で優先順位をもたせて重ね合わせることができます。この機能により、簡単なプログラムで複雑な動きの画面をコントロールすることができます。

7．通常のキャラクタ用ROM（英数字、カナ、セミグラフィックパターンなど）のほかに、ユーザーが自由に定義できるキャラクタ・ジェネレータRAMをもっており、256種のパターンを定義することができます。これらのパターンはドット単位で色を指定することができます。

8．4種類のキャラクタ表示サイズをもっており、混在して用いることができ、組合わせて使用することにより、見やすい画面を構成することができます。

9．プログラマブル・サウンドジェネレータ回路を持っており、各種の効果音を容易に作り出すことができます。

10．ディスプレイテレビCZ-800Dに接続することにより、テレビ画面とコンピュータ画面を混在して扱うことができます。

11．ディスプレイテレビCZ-800Dのチャンネル、電源、音量などをコントロールする命令をもっているため、BASICプログラム中でこれらのコントロールができます。

12．タイマー機能をもっており、会話形式で時刻設定が可能です。

13．プログラム中のジャンプ先として、ラベルを扱えるため、わかりやすいプログラムを作ることができます。

14．IF THEN ELSE, WHILE WEND, REPEAT UNTILなど構造化プログラミングに対処したステートメントを備えています。

15．専用カセットに対応した命令をいくつか用意しているので、プログラム中で自由にカセットのコントロールが行えます。

16．単精度実数の有効桁が8桁になっているため、高精度の演算を少ない容量で実現できます。

第1章 SHARP HuBASICの概要

1.1 SHARP HuBASICの起動

CZ-800Cの背面右隅にある電源スイッチをONにすると、ディスプレイテレビの画面に、

```
IPL is under preparing  ……………………………………… (1)
```

と一瞬表示されます。続いて、

```
Make ready any device
Push(F,R,C or T) key
     F:FIoppy
     R:ROM                  ………………………… (2)
     C:CMT
     T:Timer
```

と表示されて、カセットのフタが開くので、BASICカセットファイルをセットして、キーボード上のCキーを押します。すると、イニシャルローディングを行うIPL(initial program loader)が自動的にスタートして、画面に

```
IPL is looking for a program from CMT  ………………… (3)
```

と表示され、テープが回り始めます。BASICの実際のローディングが始まると、

```
IPL is loading BASIC CZ8CB01  …………………………… (4)
```

と表示されます。数分後、イニシャルローディングが終了すると、画面に

```
 ______________________________________
  SHARP-HuBASIC CZ-8CB01 VX.X
  Copyright (C) 1982 by SHARP/Hudson
 ______________________________________
                                        …… (5)
××××× Bytes free
Ok
■
```

とメッセージを表示し、カセットを自動的に巻き戻します。

Ok の下でカーソルが点滅しているのは、システムコントロールがBASICのコマンドレベルにあり、コマンドの入力待ちの状態にあることを示しています。

もし、電源投入前にあらかじめBASICカセットファイルをカセットレコーダーにセットしておくと、画面上のメッセージが(1)、(3)、(4)と連続的に変わり、IPLが自動的にスタートして、(5)のメッセージを表示してイニシャルローディングを終了します。

カセットファイルをセットするときは、テープが先頭まで巻き戻してあるかどうか必ず確かめるようにしてください。

1.2 動作モード

SHARP HuBASICを起動すると、CZ-800Cはディスプレイに Ok を表示し、入力待ちになっています。
この状態のときは、キー入力（キーボードから入力）によるすべてのBASICコマンドを受けつけます。
行番号（文番号）をつけずにステートメントを入力すると、その場で実行しますが、これをダイレクトモードにおける実行といいます。
もし、行番号をつけて入力すると、それはBASICプログラムとしてメモリーに蓄えられます。BASICの一般的記述形式は次の通りです。

行番号　ステートメント〔：ステートメント〔：ステートメント〕…〕

注）〔　〕内は省略可能

行番号は、1～65534までの整数で、プログラムの順番を示し、GOTO文、GOSUB文などのジャンプ先の番号として使います。プログラムとしてメモリーに蓄積するには、この行番号をつけて、一行入力したら ⏎ キーを押す必要があります。
行番号として0または65535を使用しても、メモリーに蓄積されません。
65536以上の数を使用した場合エラーになります。
ステートメントは、後に記述するSHARP HuBASICがもつ各種のコマンド、一般ステートメント、その他のステート、関数、および代入式のことです。
このステートメントはコロン（：）で区切ることにより、つなげて書くことができます。これをマルチステートメントと呼び、一行には、行番号も含めて、255文字まで入力可能です。

（例）

一度メモリに蓄積されたプログラムは、RUN,GOTO文、およびGOSUB文によって実行させることができます。これを、上記のダイレクトモードにおける実行に対し、プログラムモードにおける実行といいます。
⏎ キーはBASIC側へデータを与えるためのキーで命令や文の最後に必ず押して下さい！

1.3 コマンドと文

コマンドおよび文は、処理機能を表すキーワードと、処理内容を指示するオペランドから構成されます。

(例1)

(意味) プログラムの行番号を500から自動的に発生せよ。(AUTO文参照)

(例2)

(意味) 変数Iを0から100まで増分2で増やせ。(FOR文参照)

(例3)

オペランドは数値定数、文字定数、変数、関数、演算子、および区切り記号から構成されます。

1.4 定数

SHARP HuBASICで扱う定数は、数値定数と文字定数の２種類です。

1.4.1 数値定数

数値定数は正・負の数、または０です。

①整数定数

整数定数は、内部では２バイト（16ビット）で扱い、範囲が-32767～32767までの整数です。

- 10進定数として表わすには、０～９までの数字と、＋、－の符号を使いますが、＋の符号は使わなくてもかまいません。
32768以上の数に%をつけると、オーバーフローになり入力できません。

　(例)　32767
　　　　-23
　　　　-32767

- 16進定数は、最初に&Hを伴った４桁までの16進数です。16進数は、０～９までの数字と、A～Fまでの英文字を使って表します。

　(例)　&HEFD8
　　　　&H3F
　　　　&H400

- ８進定数は、最初に&Oを伴った、177777までの８進数です。８進数は、０～７までの数字を使って表します。

　(例)　&O301
　　　　&O177777
　　　　&O2350

- ２進定数は、最初に&Bを伴った、16桁までの２進数です。２進数は、０と１の２つの数字を使って表します。

　(例)　&B1010
　　　　&B1111111111111111
　　　　&B10000

②単精度定数

単精度は、有効数字８桁で表します。８桁を越える値になると、自動的に指数形式に変わります。その場合、仮数部が８桁、指数部が-38～38の範囲内になります。単精度であることを示すために、指数部はE±nとして表示されます。

　(例)　124.6
　　　　358692
　　　　-2.85E＋08

③倍精度定数

倍精度は、有効数字16桁まである数値定数で、数字の最後に#をつけて表します。16桁を越える値になると、自動的に指数形式に変わります。その場合、仮数部が16桁、指数部が-38～38の範囲内になります。倍精度であることを示すために、指数部はD±nとして表示されます。

　(例)　1192#
　　　　1.234567890123457D＋16
　　　　76895.4345265

1.4.2 文字定数

文字定数は最大255文字で、ダブルクォーテーションマーク（"）で囲まれた英数字、カナ、セミグラフィック、記号などの文字列です。ただし"は文字定数にはできません。

文字数が0の文字定数を特にヌルストリングとよび""で表します。

（例） "SHARP HuBASIC"
"1982/09/15"
"ワタシ␣ノ␣ナマエ␣ハ␣X1␣デス"

1.5 変数

変数は、英字で始まる英数字列で表わし長さは240文字以内ならば任意です。しかし、コマンドやステートメントに使われている予約語は変数名として使えません。

（例） RUNX →変数にならない。
XRUN →変数になる。

変数の型を示すために、変数のうしろに英記号が1文字伴います。

ⅰ）整数型には%をつけます。格納できる数値の範囲は、-32768～65535までの整数です。ただし、32768～65535は負の数となり-32768～-1に対応します。

（例） X%
A%
A2%

ⅱ）単精度型には！をつけますが、通常は省略してかまいません。
BASIC起動直後の変数のデフォルト値は、単精度です。
格納できる数値の範囲は、有効数字8桁、指数部-38～38のE±n表現です。

注）デフォルト値……BASICコマンド、ステートメントにおいて、パラメータを指定なしで使用したとき、パラメータに自動的にセットされる値。

（例） N
A！
D1！

ⅲ）倍精度型には#をつけます。格納できる数値の範囲は、有効数字16桁、指数部-38～38のD±n表現です。

（例） B#
PA#
M3#

ⅳ）文字型には$をつけます。格納できるのは、255文字までの文字列です。

（例） A$
XN$
X1$

（注） A、A%、A#、A$はそれぞれ区別されます。BASIC起動時はA！とAは同じです。
（DEFINT、DEFSNG、DEFDBL、DEFSTR参照）

1.6 予約語

予約語（キーワード）はBASICインタープリタが管理している語で、ユーザが変数名として使うことはできません。すべてのコマンド、ステートメント、関数、およびシステム変数が予約語になっています。
その他、下に示す語も予約語になっています。これらの予約語で始まる変数名は使えません。

AND	ATTR$	BASE	DSKF	ELSE	ERASE
EQV	FIELD	FPOS	IMP	KBUF	LOC
LOF	LSET	MOD	NAME	NOT	OR
POP	PUSH	RSET	SET	SPC	STEP
SUB	TAB	THEN	TO	TRACE	USING
VDIM	XOR				

1.7 配列変数

配列は、1つの変数名で参照できる同じ型の値の集まりです。プログラムで配列を使うときは、配列の変数名に続けて（　）の中に配列の次元と同じ個数の添字を指定します。配列の次元は255次元まで（添字は最大255個まで）許されます。
配列の大きさはDIM文で宣言しますが、宣言しなくとも配列を使うことができます。ただし宣言なしのときは添字の上限は10です。

（例）

A（x_1, x_2, …, xn）　　nは最大255
—　　xiの最大32767
↑
変数名

1.8 関数

SHARP HuBASICには、下のような関数が用意されています。

数値関数
文字関数
特殊関数
入出力用文字関数
タイマー関数

関数とは、ユーザの与えた任意の値に対し、1対1または多対1の対応で、BASIC本体が1つの値を返してくる機能のことです。実際にプログラムの文のなかでは、

関数名(引数、引数、……)

という形で変数と同じように使い、引数に数値、数式および文字列を与えると、演算処理され、その結果が関数の値になります。

(例)

```
SIN(3.14)
CHR$(76,79,86,69,76,89,33)
VAL("&H"+"EFC0")
RIGHT$("ABCDEF",3)
```

この関数はBASIC本体のなかにすでに組み込まれているので組み込み関数と呼びます。

また、BASICは上述の組み込み関数のほかに、ユーザ自身が定義できるユーザ定義関数を備えていますが、詳細はDEF FN文の項で述べます。

数値関数のほとんどは引数に倍精度の数値を与えると、関数の値を倍精度で返してきます。πは円周率を与えるシステム変数です。

1.9 式

式とは定数、変数、関数を演算子(後述)で結んだものや、単独の定数、変数、関数をいいます。

(例)

```
235
XA
"SHARP"
"&H"+A$
COS(X-Y)+.5
(X-A)*(X-B)
(A+B)/2
```

1.10 演算子

演算子には算術演算子、関係演算子、および論理演算子の3種があります。

1.10.1 算術演算子

算術演算子を下に示します。

(算術演算子)	**(演算内容)**	**(優先順位)**	**(例)**
^	累乗演算	1	X^3,X^Y
＊	乗　算	2	A＊3,A＊B
/	除　算	2	A/12,A/B(ただし、B≠0)
-	マイナス符号	3	-X
＋	加　算	4	A＋B
−	減　算	4	B - B　　A - B
¥	整数除算	5	10¥3
MOD	剰余演算	5	32.8MOD7.25 (結果は整数の割算の余りで、33÷7＝4余り5になります。)

(変数の後にMODを書く場合には、変数とMODの間にスペースをあけて下さい。)

優先順位が等しい場合は、左から先に演算されます。

演算を(　)で囲むと、かっこ中の演算が先に実行されます。

(例)

(A＋B)/2… $\frac{A+B}{2}$　　　A＋B/2…… $A+\frac{B}{2}$

1.10.2 関係演算子

関係演算子はIF文などの条件式において、2つの数値や文字列を比較するのに用います。結果は真のとき-1、偽のとき0で得られます。

下に関係演算子を示します。(IF～THEN…ELSE文参照)

(関係演算子)	**(関係演算)**	**(例)**
＝	両辺が等しい。	X＝Y
<> , ><	両辺が等しくない。	X<>Y , X><Y
<	左辺が右辺より小さい。	X<Y
>	左辺が右辺より大きい。	X>Y
<= , =<	左辺が右辺より小さいか等しい。	X<=Y , X=<Y
>= , =>	左辺が右辺より大きいか等しい。	X>=Y , X=>Y

1.10.3 文字列の比較

数値の比較と同様に、文字列も関係演算子を用いて比較することができます。

比較する文字列の短い方の長さだけ、最初から1文字づつ比較して行き、1カ所でも違っているときには、そのキャラクタコードの大きい方の文字列が大きいと判断され、最後まで同じだった場合には、長さの長い方が大きいと判断され、長さも同じの場合のみ等しいと判断されます。キャラクタコードについては後ろのコード表を参照してください。

ヌルストリング("")はどの文字列よりも小さいと判断されます。

(例)

```
""      < "A"
"AAA" > "AA"
"AB " > "AB"
"ABC" = "ABC"
```

1.10.4 論理演算子

論理演算子はいくつかの条件を判断するときに使い、論理演算をして、ビットごとに0か1を与えます。また、関係演算を2つ以上つなぐことができます。下に論理演算子を示します。

(論理演算子) **(論理演算)** **(真理値表)**

NOT 否定(not)

X	NOT X
1	0
0	1

AND 論理積(and)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR 論理和 (inclusive or)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR 排他的論理和 (exclusive or)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

IMP 包含 (implication)

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

EQV 同値 (eguivalence)

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

注)変数と論理演算子をつづけて書く場合は、間にスペースをあけて下さい。

(例)

- ●NOT　13　　13＝（0000　0000　0000　1101）$_2$
  ∴NOT13＝(1111　1111　1111　0010)$_2$
  ＝-14
- ●15AND 5　　15＝（0000　0000　0000　1111）$_2$
  5＝（0000　0000　0000　0101）$_2$
  ∴15AND 5＝（0000　0000　0000　0101）$_2$
  ＝5
- ●50 OR 44　　50＝（0000　0000　0011　0010）$_2$
  44＝（0000　0000　0010　1100）$_2$
  ∴ 50 OR 44＝（0000　0000　0011　1110）$_2$
  ＝62
- ●42XOR36　　42＝（0000　0000　0010　1010）$_2$
  36＝（0000　0000　0010　0100）$_2$
  ∴42XOR36＝（0000　0000　0000　1110）$_2$
  ＝14
- ●204 IMP136　204＝（0000　0000　1100　1100）$_2$
  136＝（0000　0000　1000　1000）$_2$
  ∴204 IMP 136＝（1111　1111　1011　1011）$_2$
  ＝-69
- ●235　EQV440　235＝（0000　0000　1110　1011）$_2$
  440＝（0000　0001　1011　1000）$_2$
  ∴235 EQV440＝（1111　1110　1010　1100）$_2$
  ＝-340

```
IF -10<X AND X<10 THEN 2000 ELSE 1000
```

(意味) Xの値が－10＜X＜10だったら2000行へ、そうでなかったら1000行へジャンプせよ。

```
IF X<-10 OR X>10 THEN 2000 ELSE 1000
```

(意味) Xの値がX＜－10またはX＞10だったら2000行へ、そうでなかったら1000行へジャンプせよ。

論理演算はプログラムではIF～THEN…ELSE…文の中で使うことが多い。

1.11 区切り記号

，(カンマ)　　PRINT,INPUT,DATAなどオペランドが並ぶ場合の区切りとして使用します。

；(セミコロン) PRINTなどの区切りとして使用します。

：(コロン)　　マルチステートメントの区切りとして使用します。

1.12 スクリーンエディタ

BASICプログラムを作成するときに、編集作業はつきものですが、これは、画面上のカーソルを使って行います。

新しいプログラムを入力するときは、先に述べたように行番号、ステートメントの順にキーボードから1文字ずつ入力して行きますが、このとき、カーソルは文字の入力に伴って1文字分ずつ画面の右方向へ移動し、画面右隅まで行くと、次の行の左隅に移動します。行番号を含めて1行につき255文字まで入力可能で、プログラムの1行は画面表示上の1行とは異なります。

プログラムを1行入力したとき画面に表示されているステートメントの文字列は、この時点では、まだBASICのメインメモリーの中に取り込まれず、テキスト画面に置かれているにすぎません。メモリーに登録するには [↵] キーを押す必要があります。以上が最も基本的なプログラムの入力方法ですが、実際には、いくつかの行からなるプログラムを入力している途中で、前のステートメントを修正したり、行番号のつけかえを行ったり、部分的な削除を行ったりするのが普通です。このような場合、カーソルによるスクリーンエディタを使って効率よくプログラムの編集を行う必要があります。

プログラムの編集はカーソルの操作方法を知ることから始まりますので、まずカーソルの移動方法を説明します。

〈カーソルの移動〉

[⇨] このキーはカーソルを右へ移動するためのもので、押し続けるとカーソルを最下行まで移動します。最下行で押し続けるとテキスト画面が1行ずつ上にスクロールします。

[⇦] このキーはカーソルを左へ移動するためのもので、押し続けるとカーソルを最上行まで移動します。

[⇧] このキーはカーソルを上へ移動するためのものです。最上行まで移動したら止まります。

[⇩] このキーはカーソルを下へ移動するためのもので、カーソルが最下行まで行ったとき、さらに押し続けるとテキスト画面が上にスクロールし始めます。

通常、カーソルキーに限らずキーボードのキーを押し続けると、同じ文字ないしは同じ運動が継続して入力されますが、REPEAT OFFと入力して [↵] キーを押すと、この機能がなくなり、1文字しか受けつけなくなります。

これはREPEAT ON [↵] キーと押すことによって解除することができます。

〈カーソルのホーム位置への移動〉

[CLR HOME] キーを押すとカーソルは画面の左上隅（これをホーム位置といいます）へ瞬時に移動します。

また、[SHIFT] を押しながら [CLR HOME] を押すと、テキスト画面がすべてクリアされカーソルはホーム位置に移動します。

〈文字の挿入〉

文字の挿入を行う場合は、[SHIFT] を押したまま [INST DEL] キーを必要な回数入力して、カーソル位置の右側に空白（スペース）を入れます。

(例)

```
10X=100
20PRINT X
```

PRINTとXの間にTAB(5);と入れたい場合は、Xのところにカーソルを持って行き、[SHIFT] を押したまま7回 [INST DEL] キーを打ってください。

```
10  X=100
20  PRINT              X
```

上図のようにPRINTとXの間が空くので、PRINTの後ろにTAB(5);と入力して [↵] キーを押すと、TAB(5);の挿入が完了します。

```
10  X=100
20  PRINT  TAB(5);X
```

〈**文字の削除**〉

文字の削除を行う場合は、 [INST DEL] を必要な回数入力して、カーソル位置の左側を削除して行きます。

(例)

```
10  X=100
20  PRINT  TAB(5);X
```

TAB(5);を削除する場合は、Xのところにカーソルを持って行き、 [INST DEL] を7回打ってください。([INST DEL] を押し続けて7回打つ分、入力することもできます。)

```
10  X=100
20  PRINTX
```

[⏎] を押すと、TAB(5);の削除が完了します。

〈**行全体の削除**〉

1つの行全体を削除する場合は、その行の行番号のみ入力して、[⏎] を押します。

(例)

```
10  X=100
20  PRINTX
10
```

上の図のように、カーソルをテキストに重ならない位置に持って行って、 [⏎] を押すと、10行目は削除されます。試しに、LIST [⏎] と押してプログラムリストを出すと、10行目が完全に削除されたことがわかります。

```
10  X=100
20  PRINTX
10
Ok
LIST
20  PRINTX   ←  10行目はなくなったのでリストされない。
Ok
```

〈**1行の挿入**〉

1つの行を挿入する場合は、挿入したい行の行番号と、ステートメントを入力して、 [⏎] を押します。

(例)

```
LIST
20  PRINTX
Ok
```

10行目にDIM A(20)と入力したい場合は、カーソルをテキストに重ならない位置に持って行って10 DIM A(20) [⏎] と押します。

```
LIST
20  PRINTX
Ok
10  DIM  A(20)
Ok
```

試しに、LIST [↵] と押して、プログラムのリストを出すと、10行目が挿入されたリストが出ます。

```
10 DIM A(20)
Ok
LIST
10 DIM A(20)
20 PRINTX
Ok
```

〈その他のスクリーンエディット機能〉

[CTRL]+[F]([CTRL]キーを押しながら[F]キーを押す)
　カーソルが1ワード分ずつ右へ移動します。

[CTRL]+[B]
　カーソルが1ワード分ずつ左へ移動します。

[CTRL]+[T]　水平TAB(タビュレーション)の設定をします。

[CTRL]+[I]　[CTRL]+[T]で設定した位置にカーソルを飛ばします。(水平TABの実行)

[CTRL]+[Y]　[CTRL]+[T]で設定した位置で、このキーを押すと、その位置だけ水平TABが解除されます。

[CTRL]+[W]　このキーを押した行と次の行とをつなぎます。

[CTRL]+[J]　カーソルの位置から右側の文字列をラインフィードして、次の行へ移します。

[CTRL]+[N]　カーソルのある行から上のテキストを上方向にスクロールします。

[CTRL]+[O]　カーソルのある行から下のテキストを下方向にスクロールします。

[CTRL]+[Z]　カーソルのある行から下のテキストをすべてクリアします。

[CTRL]+[M]　[↵]に同じ

BASICプログラムのテキストは、プログラムの実行やエラーの発生、ブレイクなどによって壊れることはありません。

ただし、行番号のついた行で[↵]を押すと、そのままメモリーに1行登録されてしまいます。

プログラムの作成過程は、プログラムを実行してエラーを発見し、スクリーンエディタによって必要な修正を行うという一連の作業から成り立つといってよいでしょう。

なお、コマンドを入力するときは[SHIFT]+[CLR HOME]キーを押して、画面をクリアしてから行うようにすると誤りが少なくてすみます。

1.13 コントロールコード

SHARP HuBASICは、通常のオペレーションのほか、さらに扱いやすいように、数多くのキー機能を用意しています。以下、キー機能を表で示します。(なお、下の表でCTRL+Aは CTRL キーを押しながら A キーを押すことを意味します)

CTRL +		出力コード	処　理　内　容	参　考
@		00		
A または	a	01	INSTモードの設定・解除をする。	
B	b	02	カーソルを1ワード分左へ戻す。	
C	c	03	実行を停止する。	SHIFT+BREAK
D	d	04	スクリーンモードなどを初期状態にする。	INIT
E	e	05	カーソルから右を行の終わりまで消す。	
F	f	06	カーソルを1ワード分右へ進める。	
G	g	07	ベルを鳴らす。	
H	h	08	文字を抹消する。	DEL
I	i	09	水平TAB(スペース出力)の実行。	HTAB
J	j	0A	ラインフィードをする。	
K	k	0B	ホームポジションへカーソルを移動する。	HOME
L	l	0C	画面を消去する。	CLR
M	m	0D	キャリッジリターンをする。	
N	n	0E	カーソル行から上を上方向へスクロールする。	
O	o	0F	〃　下を下方向へ　〃	
P	p	10		
Q	q	11	一時停止を解除する。	
R	r	12	空白を挿入する。	INS
S	s	13	一時停止をする。	BREAK
T	t	14	水平TABを設定する。	
U	u	15		
V	v	16		
W	w	17	次の行と結合する。	
X	x	18		
Y	y	19	水平TABを抹消する。	
Z	z	1A	カーソル以下の画面をすべてクリアする。	
[		1B		
¥		1C	カーソルを右へ移動	→
]		1D	カーソルを左へ移動	←
^		1E	カーソルを上へ移動	↑
_		1F	カーソルを下へ移動	↓
0			画面の背景色を黒にする。	COLOR ,0
1			〃　青　〃	〃　1
2			〃　赤　〃	〃　2
3			〃　マゼンタ　〃	〃　3
4			〃　緑　〃	〃　4
5			〃　シアン　〃	〃　5
6			〃　黄色　〃	〃　6
7			〃　白　〃	〃　7

(テンキーの) 0		文字グラフィックの色を黒にする。	COLOR 0
1		〃　青　〃	〃 1
2		〃　赤　〃	〃 2
3		〃　マゼンタ　〃	〃 3
4		〃　緑　〃	〃 4
5		〃　シアン　〃	〃 5
6		〃　黄色　〃	〃 6
7		〃　白　〃	〃 7
/		キャラクタジェネレータのROM/RAMを切り換える。	CGEN
*		文字を点滅モードにする。	CFLASH
―		文字を反転モードにする。	CREV

1.14 テレビのダイレクトコントロール

ディスプレイテレビCZ-800Dに対してのキー機能を下の表で示します。(SHIFT＋0は SHIFT キーを押しながら 0 キーを押すことを意味します。)

SHIFT＋	出力コード	処　理　内　容	参　考
(テンキーの) 0		音声ミュート。	
1		チャンネル1の設定をする。	CHANNEL 1
2		チャンネル2　〃	〃 2
3		チャンネル3　〃	〃 3
4		チャンネル4　〃	〃 4
5		チャンネル5　〃	〃 5
6		チャンネル6　〃	〃 6
7		チャンネル7　〃	〃 7
8		チャンネル8　〃	〃 8
9		チャンネル9　〃	〃 9
／		チャンネル10　〃	〃 10
＊		チャンネル11　〃	〃 11
―		チャンネル12　〃	〃 12
＋		テレビ画面とコンピュータ画面を重ねる。	CRT 2
＝		テレビ画面のみの表示。	CRT 0
・		コンピュータ画面のみの表示。	CRT 1
,		音量のノーマル設定。	
↑		音量を上げる。	
↓		音量を下げる。	
←		チャンネルの切り換え(12→11→…→2→1→12→…)	
→		〃　(1→2→…→11→12→1→…)	

第2章　コマンド・ステートメント

●コマンド・ステートメント・関数説明の形式

機　　能	各コマンド・ステートメント・関数の働きを簡潔に記述します。
書　　式	各コマンド・ステートメント・関数の書き方の一般形を示します。
省 略 形	各コマンド・ステートメント・関数の省略形を示します。
解　　説	各コマンド・ステートメント・関数の使用方法、働き、注意事項を詳しく述べています。
サンプル プログラム	使い方の実例を示し、特に説明を要するものには説明をつけています。

●コマンド・ステートメント・関数の 書式 の見方

コマンドやステートメントの 書式 の中に書かれている記号・文字には、次のような意味があります。

1．〔　〕　　〔　〕の中は任意に省略できます。
2．｛　｝　　｛　｝の中の縦に並べて書いてあるものは、任意に選択することを示しています。
3．……　　……は連続することを示します。
4．空白　　1つの空白を置くことを示します。
5．"　"　　"　"の中は文字列データであることを示します。
6．語句　　語句については書式の下で説明しています。(変数、式 etc.)

2.1 コマンド

2.1.1 CLEAR・LIMIT

機能　BASICが使用するメモリーの上限のアドレスを指定します。

書式

```
CLEAR〔アドレス〕
```

```
LIMIT アドレス
```

アドレス…使用するメモリーの上限のアドレス。&HFF00以下。

省略形　CLE.
LIM.

解説　この2つのコマンドはいずれも、メインメモリー内でBASICが使用する領域の上限のアドレスを指定するためのもので、変数、配列変数のクリアは行わず、FOR～NEXT、WHILE～WEND、REPEAT～UNTIL、GOSUB文などが格納される領域(スタック)のみクリアします。(付録「メモリーマップ」参照)

CLEARでアドレスを省略すると、全ての変数と配列変数について、数値型変数は0に、文字型変数は" "(ヌルストリング)にクリアします。これは、CLR(CLR文参照)と全く同じ機能です。

この2つのコマンドは通常、プログラムの頭に置かれ、サブルーチンの中では使用できませんので注意してください。

サンプルプログラム

```
CLEAR &HC000
OK
CLEAR
OK
LIMIT &HDF50
OK
```

2.1.2 NEWON

機能　メインメモリー内のテキストの開始アドレスを設定し、テキスト、および変数をすべて消します。

書式

```
NEWON 〔アドレス〕
```

アドレス…メインメモリー内のテキストの先頭アドレス。

省略形　なし。

解説　指定したアドレスを、メインメモリー内のテキスト(BASICプログラム)の開始アドレスとして、テキストおよび変数をすべてクリアします。アドレスを省略すると、BASIC起動時のテキストアドレスの設定となります。

サンプルプログラム

```
NEWON &HA000
OK
NEWON
OK
```

2.1.3 NEW

機　能　メモリー内のプログラムおよび変数をすべて消します。

書　式

```
NEW
```

省略形　なし。

解　説　現在メモリー内にあるテキスト(BASICプログラム)および変数をすべて消します。新しいプログラムを入れる場合には、このコマンドを実行してください。

サンプルプログラム

```
LIST
10 A=0
20 PRINT "A=";A
30 END
NEW
OK
LIST
OK
```

2.1.4 AUTO

機　　能　行番号を自動的に発生させます。

書　　式

```
AUTO ⎧m⎫ 〔,n〕
     ⎩.⎭
```

m…行番号。1～65534。
n…増　分。1～65534。

省 略 形　A.

解　　説　行番号を自動的に発生させ、プログラムの入力を行いやすいようにするためのコマンドです。mで指定される行番号から発生を開始し、増分nずつ増やしながら行番号を発生させます。このとき、行番号mを省略すると行番号10から発生を開始します。
すでに行番号つきのステートメントが入力されている場合にはそのリストが表示されますが、そのまま［⏎］を押せばリストは変更されません。
また、・（ピリオド）には、BASIC内部の持っている行管理ポインタの値が入っているので、数値と同様に扱うことができます。
なお、AUTOの中止には［SHIFT］と［BREAK］を同時に押すか［CTRL］と［C］を同時に押してください。

サンプルプログラム

```
AUTO            ← [⏎] キーを押す。
OK
 10
 20
 30             ← [SHIFT]+[BREAK] キーを押す。
OK
AUTO 100,5
OK
 100
 105
 110
 115
 120            ← [SHIFT]+[BREAK] キーを押す。
OK
AUTO 2350
OK
2350
2360
2370
2380            ← [SHIFT]+[BREAK] キーを押す。
OK
```

2.1.5 LIST・LLIST

機　能　プログラムのリストを表示します。

書　式

LIST 〔"ファイルディスクリプター:ファイル名"〔,〕〕〔〔m〕〔−〔n〕〕〕

LLIST 〔〔m〕〔−〔n〕〕〕

m………開始行番号。
n………終了行番号。

省略形　L.
LL.

解　説　開始行mより終了行nまでのプログラムリストについて、LISTは画面に、LLISTはラインプリンターに出力します。行番号を指定することによってプログラムのどこの部分でも表示させせることができます。開始行mを省略すると最初から、終了行nを省略すると最後まで出力します。CTRL と S キーを同時に押すと表示は一旦ストップし、他の任意のキーを押すと表示を再開します。

サンプルプログラム

```
10 INPUT A,B,X
20 Y=A*X+B
30 PRINT A,B,X,Y
40 END
LIST
10 INPUT A,B,X
20 Y=A*X+B
30 PRINT A,B,X,Y
40 END
OK
LIST 30
30 PRINT A,B,X,Y
OK
LIST 20-
20 Y=A*X+B
30 PRINT A,B,X,Y
40 END
OK
LIST -20
10 INPUT A,B,X
20 Y=A*X+B
OK
LIST 20-30
20 Y=A*X+B
30 PRINT A,B,X,Y
OK
```

2.1.6 RUN

機　能　プログラムの実行を始めます。

書　式

```
RUN [{n
      "ファイルディスクリプター:ファイル名"}]
```

n…行番号。

省略形　R.

解　説　行番号nからプログラムの実行を開始します。ただし、変数をすべてクリアしてから実行を開始しますので、変数をクリアせずに実行したい場合はGOTO (GOTO文参照) を使ってください。行番号nを省略すると、プログラムの最初から実行します。また、ファイルディスクリプターでファイルを直接指定すると、ロードした後、そのプログラムを即実行します。

サンプルプログラム

```
LIST
10 INPUT A,B,X
20 Y=A*X+B
30 PRINT A,B,X,Y
40 END
OK
RUN
? 2,5,4
 2            5            4            13
OK
```

2.1.7 CONT

機　　能　実行を中断したプログラムを再開します。

書　　式

```
CONT
```

省 略 形　C.

解　　説　STOP (STOP文参照) や SHIFT + BREAK でプログラムを止めた場合、CONTを実行すると、止まった次のステートメントから実行を再開します。

エラー発生時、CLEAR (CLEAR文参照) 実行直後、プログラムの書き換えなど特定条件下ではCONTが実行できずエラーが出ます。

実行再開ができるときはOk.が表示され、できないときはOk が表示されます。

サンプル プログラム

```
LIST
10 A=0
20 PRINT A
30 A=A+1
40 GOTO 20
Ok.
RUN
 0
 1
 2
 3
Break in 20     ← SHIFT + BREAK の入力
Ok.
PRINT A         ← 変数Aの内容を出力させる
 3
Ok.
A=10            ← 変数Aの内容を変える
Ok.
CONT
 11
 12
 13
 14
 15
 16
 17
```

2.1.8 EDIT

機　能　指定した行番号をリストし、カーソルを行の先頭にセットします。

書　式

EDIT (n / .)

n…行番号。

省略は・をつけるのと同じ。

省略形　E.

解　説　行番号nをリストし、nの右側にカーソルを移動させて、入力待ちになります。

エラーの発生した行の修正を能率よく行うために、このコマンドを使います。エラーが発生するとBASIC内部の・(ピリオド)にその行番号を入れるので、EDIT.(またはE..)を実行すると、エラーの発生行を直接リストして修正することができます。

サンプルプログラム

```
10 A=0
20 PLINT A
30 A=A+1
40 GOTO 20
RUN
Syntax error in 20　エラー発生
OK
EDIT.
20 █LINT A　　　　　[P]の所にカーソルが出る

20 P█INT A　　　　　[L]を[R]変えて[⏎]を押す。
LIST
10 A=0
20 PRINT A
30 A=A+1
40 GOTO 20
OK
```

2.1.9 DELETE

機能

指定した範囲のプログラムを削除します。

書式

DELETE [m | .] [{, | -}] [n | .]

m…開始行番号。

n…終了行番号。

省略形

D.

解説

開始行番号mから終了行番号nまでのプログラムを削除するためのコマンドです。プログラムの一部の行の削除は、該当する行の番号のみを入れることでできますが、まとまった行の集まりをプログラム中から削除したいときにこのコマンドを使います。なお、開始行mを省略すると、プログラムの最初から、終了行nを省略すると、プログラムの最後まで削除します。mとnの両方を省略すると、プログラム全部を削除します。

サンプルプログラム

```
LIST
10 INPUT A,B,X
20 Y=A*X+B
30 PRINT A,B,X,Y
40 END
OK
DELETE 20-30
OK
LIST
10 INPUT A,B,X
40 END
OK
```

2.1.10 RENUM

機能

プログラムの行番号を指定通り付け換えます。

書式

RENUM〔〔ℓ〕,〔m〕,〔n〕〕

ℓ…新行番号。省略すると10。

m…旧行番号。省略するとプログラムの最初の行。（ℓ≧m）

n…増分。省略すると10。

省略形

REN.

解説

旧行番号mで指定した行以降の行番号を、新行番号ℓで始まる行番号に付け換えます。新しい行番号は増分nの一定間隔で増えます。GOTO（GOTO文参照）、GOSUB（GOSUB文参照）などの飛び先の行番号も同時に変化します。
新行番号は旧行番号より大きくなければなりません。ただし、旧行番号を省略した場合は小さくてもかまいません。プログラム中で未定義の文番号は65535にセットされます。

サンプルプログラム

サンプル(1)

```
5 REM G.C.M & L.C.M
10 INPUT "A=";AI
15 IF AI<1 GOTO 130
20 INPUT "B=";BI
25 IF BI<1 GOTO 130
26 IF A<B THEN SWAP AI,BI
27 A=AI:B=BI
30 M=A MOD B
40 IF M=0 GOTO 100
50 A=B
60 B=M
70 GOTO 30
100 PRINT"G.C.M ";AI*(BI/B)
110 PRINT"L.C.M ";B
120 GOTO 10
130 END
OK
RENUM
OK
```

サンプル(2)

```
LIST
10 REM G.C.M & L.C.M
20 INPUT "A=";AI
30 IF AI<1 GOTO 160
40 INPUT "B=";BI
50 IF BI<1 GOTO 160
60 IF A<B THEN SWAP AI,BI
70 A=AI:B=BI
80 M=A MOD B
90 IF M=0 GOTO 130
100 A=B
110 B=M
120 GOTO 80
130 PRINT"G.C.M ";AI*(BI/B)
140 PRINT"L.C.M ";B
150 GOTO 20
160 END
```

2.1.11 SEARCH

機　能　指定文字列を含んでいる行のリストを出します。

書　式

```
SEARCH"x"
```

x …文字列。

省略形　SE.

解　説　プログラム中より指定文字列を探し出し、該当する行をすべてリストします。

サンプルプログラム

```
LIST
 10 REM SAMPLE
 20 INPUT A,B,X
 30 Y=A*X+B
 40 PRINT A;B;X;Y
 50 END
 OK
 SEARCH"A"
 10 REM SAMPLE
 20 INPUT A,B,X
 30 Y=A*X+B
 40 PRINT A;B;X;Y
 OK
 SEARCH" A"
 20 INPUT A,B,X
 40 PRINT A;B;X;Y
 OK
```

SEARCH "A" を実行すると、Aを含む10～40行目のリストを出力します。
SEARCH "　A" を実行すると、スペースも1文字とみるので"　A" を含む20と40行目のリストを出力します。

2.1.12 TRON・TROFF

機　　能　プログラムの実行過程の追跡と解除を行います。

書　　式

```
TRON
```

```
TROFF
```

省 略 形　TRO.

TROF.

解　　説　TRONを入力して、Ok. と表示された後、RUNを実行すると、それ以降に実行した行の番号を[　](かっこ)で囲んで、実行順に画面に表示します。

TROFFを実行すると、TRONの機能を解除します。

サンプルプログラム

```
LIST
10 REM G.C.M. & L.C.M.
20 INPUT "A=";A
30 IF A<1 THEN 20
40 INPUT "B=";B
50 IF B<1 THEN 40
60 IF A<B THEN SWAP A,B
70 X=A:Y=B
80 M=X MOD Y
90 IF M=0 THEN 130
100 X=Y:Y=M
120 GOTO 80
130 PRINT"G.C.M. ";Y
140 PRINT"L.C.M. ";A*B/Y
150 GOTO 20
OK.
TRON
OK.
RUN
[10][20]A=? 9
[30][40]B=? 12
[50][60][70][80][90][100][120][80][90][1
30]G.C.M. 3
[140]L.C.M. 36
[150][20]A=?
Break in 20        SHIFT と BREAK を同時に押す。
OK.
TROFF
OK.
RUN
A=? 9
B=? 12
G.C.M. 3
L.C.M. 36
A=?
Break in 20        SHIFT と BREAK を同時に押す。
OK.
```

2.1.13 DEVICE

機　能　デフォルトのファイルディスクリプタを決めます。

書　式

```
DEVICE "ファイルディスクリプタ:"
```

省略形　DEV.

解　説　FILES, LFILES, LOAD, LOADM, SAVE, SAVEM, LOAD ?, VERIFY, MERGE, KILL, およびRUNのそれぞれのコマンドにおいて、ファイルディスクリプタを省略すると、このコマンドによって指定されたファイルディスクリプタの指定をします。
ファイルディスクリプタを次に示します。

CRT：画面
SCR：画面
KEY：キーボード
LPT：ラインプリンター
CAS：カセットテープ
MEM：グラフィックメモリー
EMM0：外部メモリー0
　∫　　　∫
EMM9：外部メモリー9

サンプルプログラム

```
DEVICE"CRT:"
OK

DEVICE"CAS:"
OK
```

2.1.14 FILES・LFILES

機能 ファイルの一覧表を表示します。

書式

$\left\{\begin{array}{l}\text{FILES}\\\text{LFILES}\end{array}\right\}$ ["ファイルディスクリプタ:"]

ファイルディスクリプタ…CAS:, MEM:, EMM0:～EMM9:。

省略形 FIL.
LF.

解説 ファイルディスクリプタで指定したデバイス内にあるファイルの名前(ファイル名)の一覧表が、FILESのときには画面に表示され、LFILESのときにはラインプリンターにプリントされます。
ファイルディスクリプタの指定がないときは、DEVICE (DEVICE文参照)で指定されたデバイスのファイル名の一覧表が表示されます。

サンプルプログラム

```
LFILES"MEM:"
Bas       "MEM0:PROGRAM 1     .    "         '80/01/01 SUN 08:20
Bas       "MEM0:PROGRAM 2     .    "         '80/01/01 SUN 08:20
Bas       "MEM0:PROGRAM 3     .    "         '80/01/01 SUN 08:20
Bas       "MEM0:PROGRAM 4     .    "         '80/01/01 SUN 08:20
Bas       "MEM0:PROGRAM 5     .    "         '80/01/01 SUN 08:20
Bas       "MEM0:PROGRAM 6     .    "         '80/01/01 SUN 08:20
Bas       "MEM0:SAMPLE #1     .    "         '80/01/01 SUN 08:20
Bas       "MEM0:SAMPLE #2     .    "         '80/01/01 SUN 08:20
Ok.
```

2.1.15 LOAD・LOADM

機　　能　ファイルに記録されているプログラムを、メインメモリーに入れます。

書　　式

```
LOAD[[ " ファイルディスクリプタ:]ファイル名"]
```

```
LOADM[[ " ファイルディスクリプタ:]ファイル名"][,ad]
                                          [,[ad],R]
```

ad…ロード開始アドレス

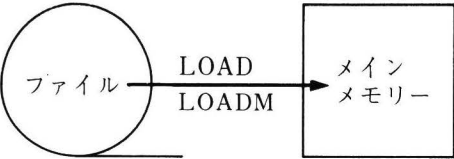
ファイルディスクリプタ…CAS:, MEM:, EMM0:～EMM9:。

省 略 形　LO.

LO.M

解　　説　BASICプログラムをメインメモリーに入れるときにはLOADを、機械語プログラムをメインメモリーに入れるときにはLOADMを実行します。

LOADにおいて、ファイル名を指定すると、そのファイルを見つけ出すまで、ほかのファイルを読み飛ばし、指定したファイルの所に来るとロードを始めます。(ファイルからメインメモリーにプログラムを移すことをロードといいます。) また、ファイル名を省略すると、最初に見つけたファイルをロードします。

LOADMにおいて、adは機械語プログラムのロードを開始するアドレスで、これを省略するとSAVEM (SAVEM文参照) で指定したセーブ開始アドレスからロードが始まります。Rオプションをつけると、ロード終了後プログラムを直ちに実行します。また、SAVEMのとき実行開始アドレスが指定されていると、LOADM終了後直ちにそのアドレスから実行を始めます。

**サンプル
プログラム**

```
LOAD
LOAD "TEST"
LOAD "CAS0:TEST"

LOADM
LOADM "TEST"
```

2.1.16 SAVE・SAVEM

機　能　メインメモリー内のプログラムをファイルに記録します。

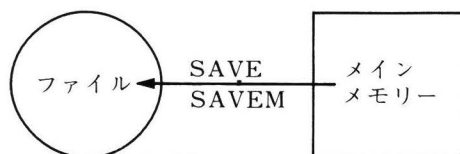
書　式

SAVE ["[ファイルディスクリプタ:] ファイル名"] [, A]

SAVEM ["[ファイルディスクリプタ:] ファイル名"] , ad_1, ad_2[, ad_3]

ad_1…セーブの開始アドレス。
ad_2…セーブの終了アドレス。
ad_3…実行開始アドレス。
ファイルディスクリプタ…CAS:, MEM:, EMM0:～EMM9:。

省略形　SA.
SA.M

解　説　SAVEを実行すると、メインメモリー内のBASICプログラムを外部ファイルに記録します。(このことをセーブするといいます。) 通常のSAVEは、バイナリィ形式にプログラムを圧縮して行いますが、Aオプションをつけると、アスキー形式で記録するので、MERGE (MERGE文参照) の実行が可能になります。アスキー形式のセーブは、リストそのままの形で行うのでバイナリィ形式よりファイルのスペースを必要としますが、外部ファイルのプログラムをMERGEするためには必ずアスキー形式でセーブしておかなければなりません。
SAVEMを実行すると、メインメモリー内の機械語プログラムを、ad_1で指定されたアドレスからad_2で指定されたアドレスまで、外部ファイルに記録します。ad_3を省略するとad_1がad_3として設定されます。

サンプルプログラム

```
SAVE
SAVE"TEST"
SAVE"CAS0:TEST"

SAVEM"CAS0:TEST",&H8000,&H80FF,&H8000
SAVEM"TEST",&H7800,&H9EFF,&H9010
```

2.1.17 LOAD？・VERIFY

機能　セーブしたプログラムが、ファイルに正しく記録されているかどうか調べます。

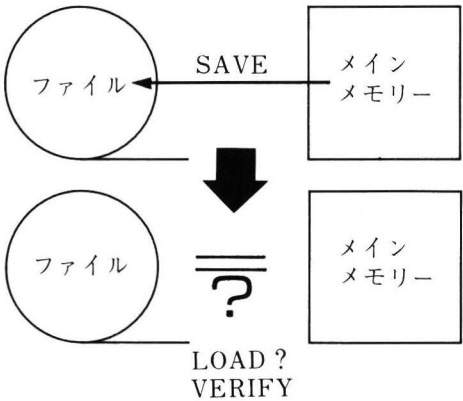
書式

LOAD？["[ファイルディスクリプタ:]ファイル名"]

VERIFY ["[ファイルディスクリプタ:]ファイル名"]

ファイルディスクリプタ…CAS:。

省略形　VE.（LOAD？の省略形はありません）

解説　LOAD？とVERIFYはまったく同じ意味のコマンドで、メインメモリー内のプログラムと、ファイル名で指定した外部ファイル内のプログラムとの照合を行います。このコマンドの実行によって、セーブしたプログラムがファイルに正しく記録されているかどうか調べることができます。また、ファイル名を省略すると、最初に見つけたファイルとメモリーとの照合を始めます。アスキー形式のファイルをVERIFY、LOAD？した場合は、ファイルの最初から最後まで空読みします。これにより、アスキー形式のファイル内の不良ブロックを見つけ出すことができます。

サンプルプログラム

```
LOAD?
LOAD?"CAS0:TEST"

VERIFY
VERIFY"TEST"
```

2.1.18 CHAIN

機能　プログラムをロードし実行します。

書式

CHAIN "[ファイルディスクリプタ:]ファイル名"

ファイルディスクリプタ…CAS:, MEM:, EMM0:～EMM9:。

省略形　CH.

解説　メインメモリー内にあるプログラムの変数を保護し、ファイルディスクリプタで指定されたファイルからプログラムをロードして実行します。

サンプルプログラム

```
CHAIN"CAS:test"

CHAIN"test"
```

2.1.19 MERGE

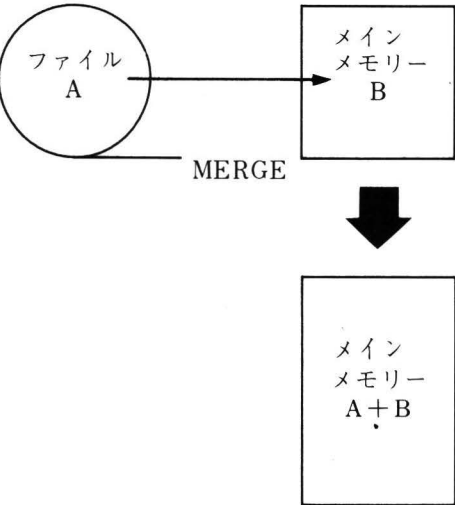
機 能　メインメモリー内のプログラムとファイル内のプログラムとを併合します。

書 式

MERGE ["[ファイルディスクリプタ:]ファイル名"]

ファイルディスクリプタ…CAS:, MEM:, EMM0:～EMM9:。

省略形　M.

解 説　ファイル名で指定したファイルからプログラムを読み込み、メインメモリー内のプログラムと組み合わせて1つのプログラムにします。このとき、双方の行番号が重複した場合は、ファイルから読み込んだ方を有効にします。ファイル内のプログラムは、アスキー形式でセーブされていなければなりません。

サンプルプログラム

```
MERGE
MERGE"TEST"
MERGE"CAS0:TEST"
```

2.1.20 KILL

機 能　ファイルに記録されたプログラムを抹消します。

書 式

KILL "[ファイルディスクリプタ:]ファイル名"

ファイルディスクリプタ… MEM:, EMM0:～EMM9:。

省略形　KI.

解 説　ファイル名で指定した外部ファイル内のプログラムを消します。ファイルディスクリプタは省略することができます。

サンプルプログラム

```
KILL "TEST"
```

2.2 一般ステートメント

2.2.1 LET

機能

式の値を変数に代入します。

書式

[LET] 変数＝ {式 / 定数 / 変数}

省略形

LETは完全に省略できます。

解説

式の演算結果を変数に代入します。左辺が数値変数のとき右辺は数値、左辺が文字列変数のとき右辺は文字列というように、両辺のタイプは一致させなければなりません。なお、LETは完全省略ステートメントなので、なくてもかまいません。

サンプルプログラム

```
10 LET A%=14
20 LET B=10.532
30 LET C!=12.234
40 LET D=12*B-C/2
50 LET E#=1.2345678901234#
60 LET F$="ABCDEFG"
```

```
10 A%=14
20 B=10.532
30 C!=12.234
40 D=12*B-C/2
50 E#=1.2345678901234#
60 F$="ABCDEFG"
```

上の2つのサンプルは全く同じ意味です。

2.2.2 DEFINT・DEFSNG・DEFDBL・DEFSTR

機能

変数の型を宣言します。

書式

```
{DEFINT} {文字1[,文字2,…]}
{DEFSNG} {文字1[-文字2]  }
{DEFDBL}
{DEFSTR}
```

文字はアルファベット1文字。

省略形

DEFI.
DEFS.
DEFD.
DEFST.

解説

この4つのDEFは、変数の型を宣言し、型の識別記号をつけなくても、宣言した型の変数として使うためのステートメントです。たとえば、Aは単精度型変数ですが、DEFSTR Aと宣言すると、文字型になり、A$とわざわざ$をつけなくても文字列の代入ができます。
この4つの型宣言は、変数の頭1文字によって行いますので、この頭文字で始まる変数は、型記号（%, !, #, $）がついていなければ、すべて宣言した型になります。
DEFINTは変数を整数型として、
DEFSNGは変数を単精度型として、
DEFDBLは変数を倍精度型として、
DEFSTRは変数を文字型として、
それぞれ定義します。
また、変数と変数の間を一（ハイフン）でつなぐと、その間に含まれるアルファベットはすべて、宣言した型になります。なお、まったく宣言しなければすべて単精度型変数になります。

サンプルプログラム

```
10 DEFINT C-H
20 DEFSTR A
30 DEFSNG I,J,K
40 DEFDBL B,L-Z
```

上のサンプルではC～Hを整数型、Aを文字型、I，J，Kを単精度型、BとL～Zを倍精度型の変数として使えるよう定義しています。

2.2.3 PRINT•LPRINT

機能

画面に情報を出力します。

書式

$$\mathrm{PRINT}[a_1]\left(\left\{\begin{matrix};\\,\end{matrix}\right\}a_2\right)\left(\left\{\begin{matrix};\\,\end{matrix}\right\}a_3\right)\cdots\cdots$$

$$\mathrm{LPRINT}[a_1]\left(\left\{\begin{matrix};\\,\end{matrix}\right\}a_2\right)\left(\left\{\begin{matrix};\\,\end{matrix}\right\}a_3\right)\cdots\cdots$$

a_1, a_2, a_3……式、変数、定数。

省略形

?またはP.

LP.

解説

PRINTは指定した式の値、文字列、変数、定数の値を画面に表示します。PRINTのみのときは1行改行し、式、変数、定数の間や最後に;(セミコロン)をつければ、改行せずに続けて表示することができます。LPRINTは指定した式の値、文字列、変数、定数の値をラインプリンターにプリントします。改行についてはPRINTと同様です。

なお、?(クエスチョンマーク)をPRINTと同じステートメントとして使うことができます。

サンプルプログラム

```
LIST
10 REM SAMPLE
20 A=15
30 B=-30
40 C$="HELLO!"
50 D$="Hi!"
60 PRINT TAB(5);A;C$
70 PRINT B;TAB(5);D$
80 PRINT B;SPC(5);D$
90 PRINT "ABCDEFGHIJK";22,"SHARP X-1";-82
OK
RUN
        15 HELLO!
-30  Hi!
-30       Hi!
ABCDEFGHIJK 22         SHARP X-1-82
OK
```

TAB(n)は画面の左端からn文字分飛ぶという意味で、SPC(n)は位置に関係なくn文字の空白を書くという意味です。PRINTの書式(USINGの使い方)については書式指定の章(PRINT USING)で詳述します。

2.2.4 WRITE

機 能 画面にデータを表示します。

書 式

WRITE [a_1] $\left[\left\{ \begin{matrix} , \\ ; \end{matrix} \right\} a_2 \right] \left[\left\{ \begin{matrix} , \\ ; \end{matrix} \right\} a_3 \right]$……

a_1, a_2, a_3……式、変数、定数。

省略形 WR.

解 説 画面に式の値を ,(カンマ)で区切り、詰めて表示します。文字型のときは"で囲んで表示します。

サンプルプログラム

```
WRITE 1,-2,"123"
1,-2,"123"
OK
WRITE 1;-2;"123"
1,-2,"123"
OK
WRITE

OK
```

2.2.5 INPUT

機 能 キーボードからデータを入力します。

書 式

INPUT ["文字列"] $\left(\begin{matrix} ; \\ , \end{matrix} \right)$ 変数1 [, 変数2,……]

文字列……画面に表示する文字列。

変　数……キーボードから入力する変数。

省略形 I.

解 説 キーボードから入力した数値や文字を変数に入れます。 "(ダブルクォーテーションマーク) で文字列を囲むことによって、ステートメントの実行時に、その文字列を画面に表示し、キーボードからの入力を待ちます。このとき、文字列と変数が ;(セミコロン) で区切られていると文字列の後ろに? (クエスチョンマーク) がつき ,(カンマ) で区切られていると?は表示しません。キーボードからデータを入力して、 ⏎ キーを押すとそのデータを変数に入れます。
変数の型とキーボードから入力する変数の型は一致していなければなりません。複数個の変数を「,」で区切って指定しているときには、その指定順に、対応するデータを「,」で区切って一度に入力します。
何も入力しないで ⏎ キーを押すと変数の値を保存して次のステートメントに進みます。

サンプルプログラム

```
10 INPUT "A=";A
20 INPUT "B,C,D=";B,C,D
30 PRINT "A=";A
40 PRINT "B,C,D=";B;C;D
50 END
RUN
A=? 5                        ←5を入力
B,C,D=? 5,15,3.14            ←5と15と3.14を入力
A= 5
B,C,D= 5  15  3.14
OK
```

2.2.6 LINPUT・LINE INPUT

機　能　キーボードから文字を入力します。ブランク（空白）やカンマ（,）も入力できます。

書　式

LINPUT ["文字列" {; ,}] 文字列変数

LINE INPUT ["文字列" {; ,}] 文字列変数

文字列……画面に表示する文字列。
文字列変数……キーボードから入力する文字変数。

省略形　LI.
LINEI.

解　説　キーボードから入力した文字列を文字列変数に入れます。このとき ,（カンマ）や空白（スペース）も1文字として入力できます。INPUT（INPUT文参照）と同様に "（ダブルクォーテーションマーク）で囲った文字列を画面に表示できますが、LINPUTでは、これも同時に入力してしまいますので、プログラム中で取り除く処理が必要です。なお , と ;（セミコロン）のどちらを使っても同じ結果となります。入力できる最大文字数は255文字までです。

サンプルプログラム

```
LINEINPUT"STR=";A$
STR=123,456,"""""",,,,
OK
PRINT A$
STR=123,456,"""""",,,,
OK
```

2.2.7 CLEAR・CLR

機　能　変数および配列をすべてクリアします。

書　式

CLEAR

CLR

省略形　CLE.（CLRにはありません）

解　説　すべての変数および配列のうち数値型のものを0に、文字型のものをヌルストリング（何も入ってない状態）にします。

サンプルプログラム

```
A=1000:B$="123"
OK
?A,B$;"*"
 1000       123*
OK
CLEAR
OK
?A,B$;"*"
 0          *
OK
```

2.2.8 OPTION BASE

機　　能　配列の添字の下限を宣言します。

書　　式

```
OPTION BASE n
```

n＝0, 1。

省 略 形　OP. B.

解　　説　配列の添字は通常下限値が0になっていますが、このステートメントによって、1にすることができます。配列のディメンジョンが切られる前に一度だけ宣言でき、再宣言できません。
CLR, CLEAR, RUN, NEW [ON] によって、この宣言は解除されます。

サンプルプログラム

```
NEW
OK
OPTION BASE 1
OK
DIM A(10)
OK
OPTION BASE 0
Duplicate Definition
OK
```

2.2.9 DIM

機　　能　配列変数を定義します。

書　　式

```
DIM 配列名1 (m1 [, m2 , ……] ) [ , 配列名2 (n1 [ , n2 , …] ) ] ……
```

配列名……配列型変数名。

m_1, m_2, n_1, n_2 …… : 添字の上限。

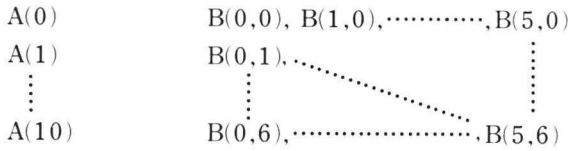
省 略 形　DI.

解　　説　配列変数の名前とその添字の上限を設定します。1つのDIMで複数の配列変数を定義でき、配列ごとに、メモリー範囲内で255個の添字（255次元）の指定ができます。(一行の入力文字数の制限から実際には255次元までは指定できません。) DIMの実行後、数値型配列には0が入り、文字型配列はヌルストリングになります。配列添字の下限は、通常は0ですがOPTION BASE 1を宣言すると、1に変更することができます。

サンプルプログラム

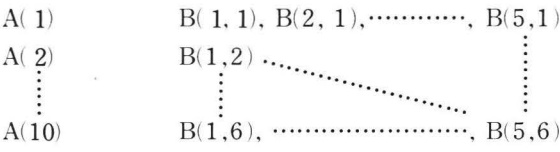
```
10 OPTION BASE 0
20 DIN A(10),B(5,6)
```

上の例では、具体的には

```
A(0)     B(0,0), B(1,0),…………,B(5,0)
A(1)     B(0,1),
 :         :                      :
A(10)    B(0,6),……………………,B(5,6)
```

というような2つの配列が定義され、すべての配列要素に0が入ります。

```
10 OPTION BASE 1
20 DIN A(10),B(5,6)
```

OPTION BASE 1のときは、

```
A( 1)    B( 1, 1), B(2, 1),…………, B(5,1)
A( 2)    B(1,2)
 :         :                      :
A(10)    B(1,6), ……………………, B(5,6)
```

というような2つの配列が定義され、すべての配列要素には0が入ります。

2.2.10 LABEL

機能　プログラム中にラベルをつけます。

書式

```
LABEL "ラベル名"
```

ラベル名……255文字までの文字列。

省略形　LA.

解説　プログラム中にラベルをつけます。このラベルは、GOTO, GOSUBなどのジャンプ先として参照される目印で、これによってプログラムにドキュメント性をもたせることができます。ラベル名の制限は特にありません。プログラム中に同じラベルがある場合は最初に出てきたラベルが優先されます。

サンプルプログラム

```
10 A=0
20 LABEL"ｲﾝｻﾂ":PRINT A;
30 A=A+1
40 GOTO "ｲﾝｻﾂ"
```

2.2.11 GOTO・GO TO

機能　指定した行番号かラベルへ無条件にジャンプします。

書式

```
{GOTO  } {行番号    }
{GO  TO} {"ラベル名"}
```

行番号……ジャンプ先の行番号。
ラベル名……LABELで定義された文字列。

省略形　G.

解説　GOTOとGO TOは同じ機能を持ち、指定した行番号かラベルへ、無条件にジャンプするもので、プログラムの流れを変えるときに使用します。ラベルはLABEL（LABEL文参照）によってエントリーされた文字列です。

サンプルプログラム

```
10 A=0
20 LABEL"ｲﾝｻﾂ"PRINT A;
30 A=A+1
40 GOTO "ｲﾝｻﾂ"

10 A=0
20 PRINT A;
30 A=A+1
40 GOTO 20
```

上のサンプルはどちらも同じ意味です。
止めるときは、SHIFT＋BREAKを押してください。

2.2.12 GOSUB・GO SUB

機能 BASICプログラム内のサブルーチンを呼び出します。

書式

```
{GOSUB  }{ 行番号   }
{GO  SUB}{"ラベル名"}
```

行番号……サブルーチンの開始行番号。

ラベル名……LABELで定義された文字列。

省略形 GOS.

解説 メモリー中にあるBASICのサブルーチンを呼び出し、指定された行番号またはラベル名からRETURN（RETURN文参照）までを実行させます。呼び出されるサブルーチンの最後には必ずRETURNがなければなりません。

サンプルプログラム

```
10 INPUT"A,B=";A,B
20 GOSUB 100
30 PRINT "C=";C
40 END
100 C=(A-B)/2
110 RETURN
RUN
A,B=? 3,12
C=-4.5
OK
```

2.2.13 RETURN

機能 GOSUBで呼ばれたサブルーチンの最後につけて戻りに使用します。

書式

```
RETURN ({ 行番号   })
        {"ラベル名"}
```

行番号……サブルーチンからの戻り先行番号。

ラベル名……LABELで定義された文字列。

省略形 RE.

解説 RETURNはGOSUBで呼ばれたサブルーチンから戻り先へ戻るときに使用します。行番号、ラベル名をつけると、その行へ戻りますが、省略するとGOSUB文の後ろに戻ります。

サンプルプログラム GOSUB文参照。

```
RETURN
RETURN 100
RETURN "MAIN PROG"
```

2.2.14 IF～THEN……ELSE

機能　理論式を判断して次に進みます。

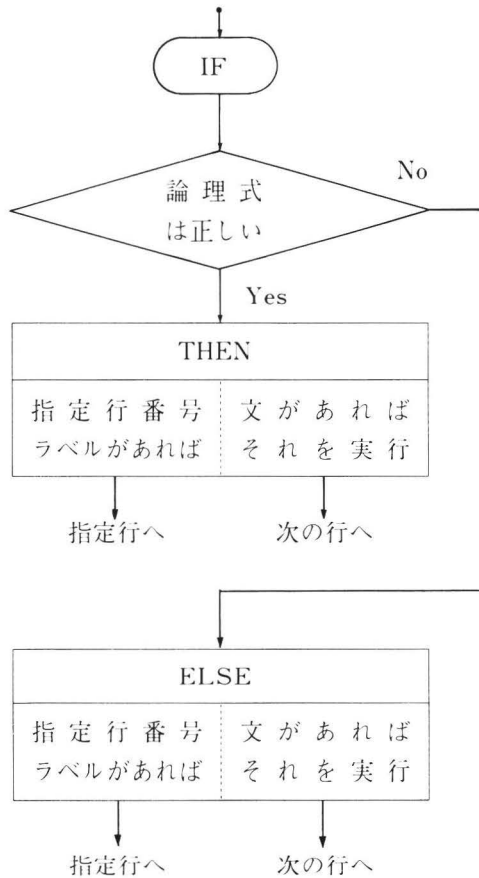
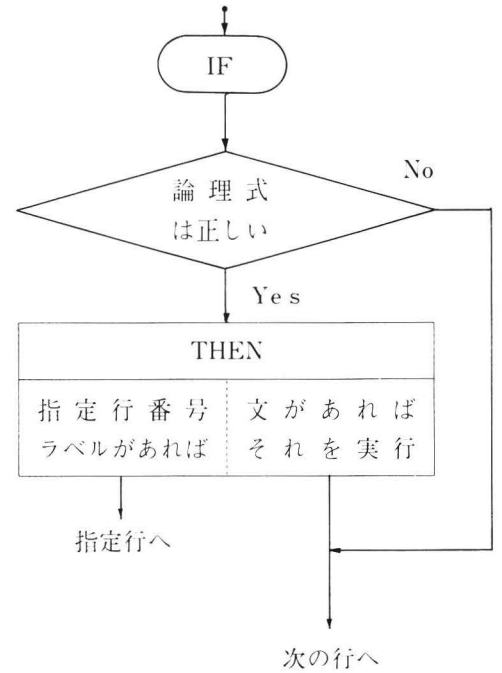
書式

行番号……ジャンプ先の行番号。

ラベル名……LABEL（LABEL文参照）で定義された文字列。

文……………任意のステートメント。

省略形　IF～TH.……EL.……

解説　論理式が正しいかどうか判断して、指定の行番号やステートメントに制御を移して実行を続けます。
ELSE以降がある場合、論理式が正しければ、THENの後ろで指定された行番号・ラベルの行、または文の実行に移り、まちがっていれば、ELSEの後ろで指定された行番号・ラベルの行、または文の実行に移ります。
ELSE以降がない場合、論理式が正しければ、THENの後ろで指定された行番号・ラベルの行、または文の実行に移り、まちがっていれば、次の行へ移ります。
THENとELSEの後ろの文として、IF文を使ってもかまいません。ただし、この場合は最後のIF文のELSEのみ省略することができます。なお、THENおよびELSEの前は1文字分スペースをあけてください。次に、このステートメントの流れ図を示します。

サンプル
プログラム

```
LIST
10 INPUT A
20 IF A=0 THEN PRINT"A=0" ELSE PRINT"A<>0"
OK
RUN
? 1
A<>0
OK
RUN
? 0
A=0
OK

LIST
10 INPUT A,B
20 IF A=0 THEN IF B=0 THEN PRINT"A=0,B=0" ELSE PRINT"A=0,B<>0"
ELSE IF B=0 THEN PRINT"A<>0,B=0" ELSE PRINT"A<>0,B<>0"
30 GOTO 10
OK
RUN
? 0,0
A=0,B=0
? 0,1
A=0,B<>0
? 1,0
A<>0,B=0
? 1,1
A<>0,B<>0
?
Break in 10 ← SHIFT BREAK を押す
Ok.
```

2.2.15 FOR～TO…STEP…

機　能　FORとNEXTとの間を繰返し実行します。

書　式

```
FOR 変数名＝初期値 TO 終了値［STEP 増分］
```

変数名……単精度型変数、整数型変数。

初期値、終了値、増分……定数、変数、式。負の数でもよい。

省略形　F.

解　説　FORはNEXTとともに用いられ、変数の値が終了値からはみ出すまでFORからNEXTへの実行を続け、はみ出すとNEXTの次へ実行を移します。変数は初期値の値から始まって、この文が実行されるたびに増分だけ加えられて行き、終了値をはみ出すと繰返しが終わります。なお、STEP増分を省略すると増分は1になります。

対応するNEXTがない場合はFOR without NEXTエラーが出ます。変数の初期値が終了条件をすでに満たしている場合は、そのままNEXTの次へ実行を移します。TOとSTEPの前には1文字分スペースをあけてください。

サンプルプログラム

```
LIST
10 FOR I=1 TO 4
20 FOR J=1 TO I
30 FOR K=J TO I
40 PRINT"*";
50 NEXT K
60 PRINT
70 NEXT J
80 PRINT
90 NEXT I
Ok
RUN
*

**
*

***
**
*

****
***
**
*

Ok
```

注）NEXTの後ろの変数を省略するほうが実行速度が上がりますが、ここではわかりやすくするため、変数をつけています。

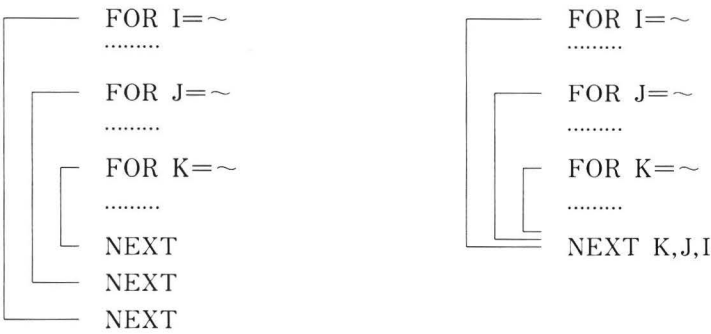
2.2.16 NEXT

機能　FORの終端を示します。

書式

```
NEXT [変数, 変数……]
```

省略形　N.

解説　NEXTの変数は、FORの変数と1対1に対応していなければなりません。FORとNEXTの中にさらに別のFORとNEXTを置くことができますが、これを入れ子の構造といい、入れ子は何重でもかまいません。

どちらも同じ構造です。

変数を省略するほうがループ速度が速くなります。

FOR文参照。

サンプルプログラム

```
NEXT
NEXT I
NEXT C,B,A
```

2.2.17 REPEAT

機能　REPEAT～UNTILの初めを示します。

書式

```
REPEAT
```

省略形　REP.

解説　REPEATは、UNTILと対にして使用し、ループの初めを示すのみです。

対応するUNTILがない場合はエラーになります。

(UNTIL文参照)

UNTIL文参照。

サンプルプログラム

```
REPEAT
```

2.2.18 UNTIL

機　能　REPEATとUNTILとの間を繰返し実行します。

書　式

```
REPEAT
  :
  :
UNTIL 論理式
```

省略形　U.

解　説　このステートメントはREPEATループの端末で、論理式がまちがっていればREPEATへ戻り、正しければ次の行へ通り抜けます。
なお、REPEAT〜UNTILの間にGOTO文を入れてループの外にジャンプさせることはできません。
初期値を入力すると、ループを回るたびに1ずつ減じて行き、1より小さくなると次のPRINT文に移ります。

サンプルプログラム

```
10 INPUT"ｼｮｷﾁ=";A
20 REPEAT
30 PRINT A
40 A=A-1
50 UNTIL A<1
60 PRINT "ｵﾜﾘ"
RUN
ｼｮｷﾁ=? 5
 5
 4
 3
 2
 1
ｵﾜﾘ
OK
RUN
ｼｮｷﾁ=? -5
 -5
ｵﾜﾘ
OK
```

2.2.19 WHILE

機　能

WHILEとWENDとの間を繰返し実行します。

書　式

```
WHILE 論理式
  :
  :
WEND
```

省略形

W.

解　説

WHILEはWENDとともに用いられ、論理式が正しければWHILEとWENDとの間の実行を続け、まちがっているときWENDの次の文へジャンプします。注意するのは、論理式が最初で判断されている点で、場合によっては、WHILEとWENDの中を一度も通らないで次にジャンプすることもあります。ここに、REPEAT～UNTILとの相違点があります。WHILEとWENDが正しく対応していない場合エラーになりままず。なお、WHILE～WENDの間にGOTO文を入れてループの外にジャンプさせることはできません。

サンプルプログラム

INPUT文で入力した初期値が1以上であれば、出力して1減じますが、1より小さくなると60行目のPRINT文にジャンプします。

```
10 INPUT"ショキチ=";A
20 WHILE A>=1
30 PRINT A
40 A=A-1
50 WEND
60 PRINT "オワリ"
RUN
ショキチ=? 5
 5
 4
 3
 2
 1
オワリ
OK
RUN
ショキチ=? -5
オワリ
OK
```

2.2.20 WEND

機　能

WHILEの終端を示します。

書　式

```
WEND
```

省略形

WE.

解　説

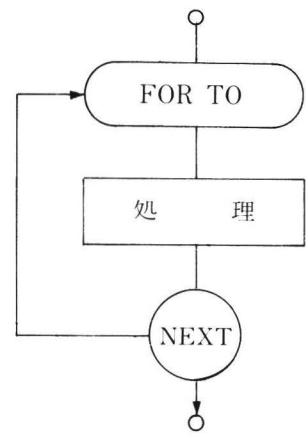
このステートメントはWHILEの終端として使用されるのみです。
（WHILE文参照）

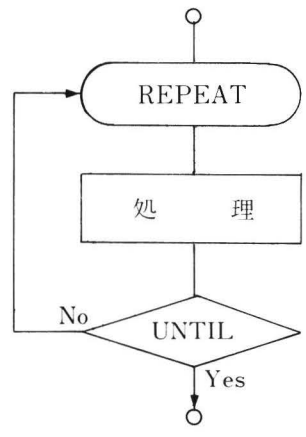
サンプルプログラム

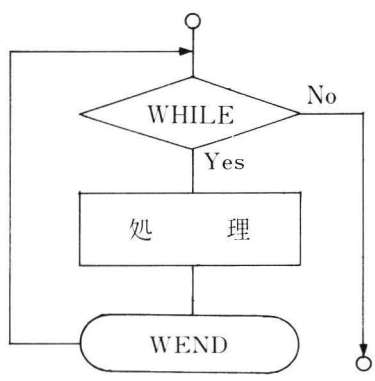
WHILE文参照。

```
WEND
```

〈ループ文のフローチャート〉

(1) FOR NEXT ループ (FOR文参照)

FOR i=m TO n STEP s
mからnまでsずつ増えながら、NEXTとの間でループします。
mの値がnをはみ出すときにループが終了します。
ループに入る前から、mの値がnをはみ出している時は1回もループを通りません。

(2)REPEAT UNTIL ループ (UNTIL文参照)

REPEATとUNTILとの間でループします。論理式は終端のUNTILに記述し、真の値をとるまでループします。よって、少なくとも1回はループを通ることになります。

(3) WHILE WEND ループ (WHILE文参照)

WHILEとWENDとの間でループします。論理式は入口のWHILEに記述し、真の値をとっている間ループします。よって、1回もループを通らない場合もあり得ます。

2.2.21 ON～

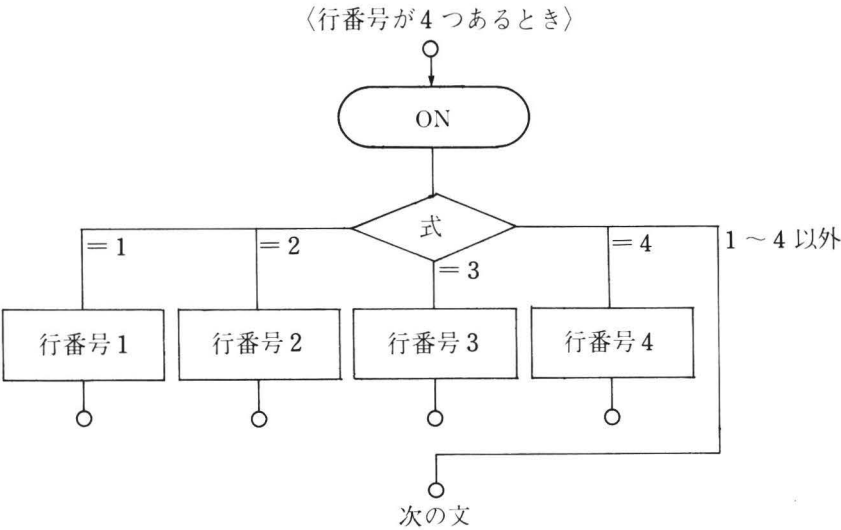
機　　能　式の値によって、いくつかの行へジャンプします。

書　　式

```
ON 式 ┌GOTO        行番号1 [, 行番号2, 行番号3,……] ┐
      │GOSUB              〃                     │
      ┤RETURN             〃                     ├
      │RESTORE            〃                     │
      └RESUME             〃                     ┘
```

　行番号のかわりにラベル名も使えます。式の後ろには必ず空白を1つ以上置いてください。

省 略 形　O.

解　　説　式の値（数値）がiのとき、i番目に指定した行番号の行にジャンプして、それ以降の行を実行します。式の値が1ならば1番目の行番号、2ならば2番目、3ならば3番目、……というように対応しており、式の値に対応する行番号がなければON文の次を実行します。なお式の値が整数でなければ、小数第1位で四捨五入された値になります。

サンプル
プログラム

```
LIST
5 REM SAMPLE PROGRAM
10 PRINT "1:ﾀｼｻﾞﾝ"
20 PRINT "2:ﾋｷｻﾞﾝ"
30 PRINT "3:ｶｹｻﾞﾝ"
40 PRINT "4:ﾜﾘｻﾞﾝ"
50 INPUT "1-4?",A:IF A<1 GOTO 50
60 IF A>4 GOTO 50
70 ON A GOSUB 100,200,300,400
80 PRINT "ｺﾀｴ";C
90 GOTO 10
100 GOSUB 500
110 C=A+B
120 RETURN
200 GOSUB 500
210 C=A-B
220 RETURN
300 GOSUB 500
310 C=A*B
320 RETURN
400 GOSUB 500
410 IF B=0 THEN PRINT"ｺﾀｴﾊ ｱﾘﾏｾﾝ!":RETUR
N 10
420 C=A/B
430 RETURN
500 INPUT "ｶｽﾞｦ2ﾂｲﾚﾃ",A,B
510 RETURN
OK
```

2.2.22 STOP

機　能　プログラムの実行をストップします。

書　式

```
STOP
```

省略形　S.

解　説　プログラム中にSTOPがあると、「Break in 行番号」を画面に表示し、止まります（行番号はSTOPのあった行を示しています）。プログラムの停止後にCONT（CONT文参照）を実行すると、止まった次の文から実行を再開します。

サンプルプログラム

```
LIST
10 PRINT"No 10"
20 STOP
30 PRINT"No 30"
OK
RUN
No 10
Break in 20
Ok.
CONT
No 30
OK
```

2.2.23 END

機　能　プログラムの実行の終了を宣言します。

書　式

```
END
```

省略形　EN.

解　説　このステートメントを実行するとプログラムは終了し、すべてのファイルを閉じて、画面上にはOkが表示されて入力待ちとなります。

サンプルプログラム

```
LIST
10 PRINT"No 10"
30 PRINT"No 30"
OK
RUN
No 10
No 30
OK
20 END
LIST
10 PRINT"No 10"
20 END
30 PRINT"No 30"
OK
RUN
No 10
OK
```

2.2.24 SWAP

機　　能　2つの変数の値を交換します。

書　　式

```
SWAP 変数1, 変数2
```

変数…値を変換する変数。

省 略 形　SW.

解　　説　変数1の値を変数2に、変数2の値を変数1に入れて、値の交換を行います。変数は数値、文字型いずれでもかまいませんが、2つの変数の型は一致していなければなりません。

サンプルプログラム

```
10 REM SAMPLE PROGRAM
20 A=10
30 B=-9
40 C$="CC"
50 D$="DD"
60 E$="EE"
70 PRINT A;B
80 PRINT C$;D$;E$
90 SWAP A,B
100 SWAP C$,D$:SWAP D$,E$
110 PRINT A;B
120 PRINT C$;D$;E$
130 END
OK
RUN
 10 -9
CCDDEE
-9  10
DDEECC
OK
```

2.2.25 REM

プログラム文にコメントを入れます。

機　　能

```
REM [コメント]
```

書　　式　コメント……任意の文字列

省 略 形　'(アポストロフィ)

解　　説　プログラムにコメントを挿入するためのステートメントで、プログラムの実行にはかかわりありません。REMは '(アポストロフィ) で代用できます。

サンプルプログラム

```
10 REM Sample program
15 REM
20 REM SHARP/Sharp/ｼｬｰﾌﾟ
30 REM
40 REM 「REM」 ﾊ ﾌﾟﾛｸﾞﾗﾑ ﾆ ｺﾒﾝﾄ ｦ ｲﾚﾙﾀﾒ ﾉ
50 REM  ｽﾃｰﾄﾒﾝﾄ ﾃﾞｽ｡
60 '
70 '       「'」=「REM」
80 '
```

2.2.26 READ

機能　DATA文で用意されたデータを変数に入れます。

書式

READ 変数1 [, 変数2 , 変数3 ,……]

変数……DATA文のデータを入れる変数

省略形　REA.

解説　DATA文で用意されたデータを読み込んで、変数に割り当てます。READ文はいつもDATA文と対にして使い、DATA文の定数データは、READ文の変数と1対1に対応し、かつ双方同じ型でなければなりません。DATA文は行番号の若い順にデータの並びの先頭から読み込まれ、DATA 文の定数データの数がREAD 文の変数の数より多いときには、次のREAD文により引き続き読み込まれます。
データの数が不足しているときは、Out of data エラーになります。RESTORE（RESTORE文参照）を使うと、DATA文を読み直したり、DATA文の読み込み順を変えることができます。

70行目の数値データを20行目で読み込み、Xに入れて30行目のPRINT文で出力します。続いて、80行目の文字列データを50行目で読み込み、A$, B$, C$に入れて60, 65行目のPRINT文で出力します。

サンプルプログラム

```
LIST
10 FOR I=1 TO 7
20 READ X
30 PRINT X;
40 NEXT
50 PRINT
60 READ A$,B$,C$
70 PRINT A$;" ";B$
80 PRINT A$;" ";C$
90 DATA 8,0,1.2,-4,&H1A,&B101,12
100 DATA Good,morning,evening
OK
RUN
 8  0  1.2 -4  26  5  12
Good morning
Good evening
OK
```

2.2.27 DATA

機　能　READで読むデータを定義します。

書　式

```
DATA 定数1[,定数2,……]
```

定数……数値定数、文字列のデータ。

省略形　DA.

解　説　READ文で読み込むデータを用意するステートメントで、何かを実行するというステートメントではないので、プログラム中のどこにでも、いくつでも置くことができます。1つのDATA文によって用意できるデータは、最大255文字分の数値定数および文字列定数です。文字列をデータとして用意する場合、"(ダブルコート)で囲まなくても使用できますが ,(カンマ)を区切りに使うので、カンマをデータにする場合にはダブルコートで囲まなければなりません。また、:(コロン)をデータにする場合にもダブルコートで囲む必要があります。

サンプルプログラム

```
1000 DATA 23,43,55,65,12,54,34,54,99
1010 DATA 0,33,67,93,10,20,55,77,53,45

1500 DATA Suzuki,Tanaka,Abe,Yamada,Kubo
1510 DATA Nomura,Chiba,Uchida,Kimura

2000 DATA "A,A","A,B","A,C","B,A","B,B"
2010 DATA "B,C","C,A","C,B","C,C"
```

1000と1010行は数値データ、
1500と1510行は文字列データ、
2000と2010行は文字列データにカンマが入った場合
をそれぞれ示しています。
なお、数値データと文字列データを合わせて、定数データと呼ぶことがあります。

2.2.28 RESTORE

機　　能　READ文で読み始めるDATA文を指定します。

書　　式

RESTORE［｛行番号 / "ラベル名"｝］

行番号……読み始めるDATA文の行番号。
"ラベル名" ……LABELで指定した文字列。

省 略 形　RES.

解　　説　このステートメント実行後のREAD文は、行番号、"ラベル名" で指定したDATA文のデータから読み込みを開始し、RESTOREのみのときは、プログラム中最初に現れるDATA文から読み込みを開始します。
20行目で1020行からデータを読み込むよう指定し、80行目で "データ" というラベル名のところからデータを読み込むよう指定しています。

サンプル プログラム

```
LIST
10 REM
20 RESTORE 1020
30 FOR I=1 TO 5
40 READ A
50 PRINT A;
60 NEXT
70 PRINT
80 RESTORE "ﾃﾞｰﾀ"
90 FOR I=1 TO 5
100 READ A
110 PRINT A;
120 NEXT
1000 '
1010 LABEL "ﾃﾞｰﾀ":DATA 23,43,55,65,12
1020 DATA 12,56,34,78,56
OK
RUN
 12  56  34  78  56
 23  43  55  65  12
OK
```

2.2.29 DEF FN

機　　能　ユーザーが関数を定義するのに使います。

書　　式

DEF FN関数名 $(x_1[,x_2, \cdots\cdots])$= 式

関数名…FNに続けて書きます。数値型、文字型のどちらでも可能です。

x_1, x_2, …パラメータ。

省 略 形　なし。

解　　説

ユーザーの作った関数を自由に定義することができ、プログラム中で使うことができます。関数名と定義式の型が一致していれば、文字列関数も定義できます。

3つの関数

$$FNA(X, Y, Z) = \frac{X+Y+Z}{3}$$

$$FNB(X, Y, Z) = \sqrt[3]{XYZ}$$

$$FNC(X, Y, Z) = \frac{XYZ}{3}$$

を定義して、X, Y, Zに任意の値を代入して関数の値を計算してみましょう。

サンプルプログラム

```
LIST
10 DEF FNA(X,Y,Z)=(X+Y+Z)/3
20 DEF FNB(X,Y,Z)=(X*Y*Z)^(1/3)
30 DEF FNC(X,Y,Z)=(X*Y*Z)/3
40 INPUT"D=";D:INPUT"E=";E:INPUT"F=";F
50 PRINT "FNA=";FNA(D,E,F),"FNB=";FNB(D,E,F),"FNC=";FNC(D,E,F
60 PRINT
70 PRINTX,Y,Z
OK
RUN
D=? 1
E=? 2
F=? 3
FNA= 2      FNB= 1.8171206       FNC= 2

 0          0          0
OK
```

X, Y, Zの各変数は
影響を受けません。

2.2.30 DEF USR

機　能　ユーザーが機械語サブルーチンを定義するのに使います。

書　式

```
DEF USR[番号]＝アドレス
```

番号………機械語サブルーチンの番号。0～9の整数で、省略すると0。
アドレス…機械語サブルーチンの先頭のアドレス。

省略形　なし。

解　説

ユーザーが機械語で作った機械語サブーチンを呼び出すUSR関数に番号をつけその実行開始の先頭アドレスを設定します。USR関数は、0～9まで登録テーブルをBASIC内部に持っていて、最大10まで関数を定義することができます。いくつかのUSR関数の識別にこの0～9の番号を使いますが、同じ番号のUSR関数を何度でも設定しなおすことができます。

機械語サブルーチンの使い方については付録「USR命令の使い方」を参照してください。

サンプルプログラム

```
LIST
10 CLEAR &HFE00
20 POKE &HFE00,&HC9
30 DEF USR1=&HFE00
40 PRINT USR1(10)
OK
RUN
 10
OK
```

2.2.31 CALL

機　能　機械語サブルーチンを直接呼び出します。

書　式

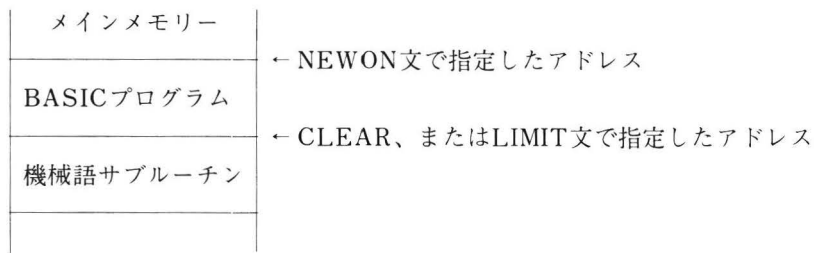
```
CALL アドレス
```

アドレス…サブルーチンの実行開始番地

省略形　CA.

解　説

アドレスを実行開始番地とする機械語サブルーチンを呼び出します。
アドレスには16進定数（&HXXXXの形）、10進整定数（—32768～32767）などの整数定数、整数変数、式が指定されます。

機械語からBASICへ戻るには機械語命令のRET（&HC9）を実行させなければなりません。
機械語サブルーチンは、CLEARまたはLIMIT文で指定したアドレス以降に置いて下さい。

サンプルプログラム

```
LIST
10 CLEAR &HFE00
20 POKE &HFE00,&HC9
30 CALL &HFE00
OK
RUN
OK
```

2.2.32 POKE

機　　能　メインメモリー内の指定アドレスにデータを書き込みます。(⇐⇒PEEK)

書　　式

```
POKE アドレス, データ[,データ,データ,…]
```

アドレス…データの書き込みアドレス。0～&HFFFF。
データ……0～255までの整数

省 略 形　PO.

解　　説　メインメモリー内の指定したアドレスに1バイト（8ビット）のデータを書き込みます。
データを,（カンマ）で区切って続けて書くと、連続したアドレスに書き込むことができます。
POKEの使用に際しては、BASICプログラムなどメインメモリー内のシステムを破壊してしまう恐れがあるので十分注意してください。

サンプルプログラム

```
10 CLEAR &HBFFF:DT=0
20 FOR AD=&HC000 TO &HC01F
30 POKE AD,DT              'メモリー ヘノ カキコミ。
40 DT=DT+1
50 NEXT
60 FOR AD=&HC000 TO &HC01F
70 DT=PEEK(AD)             'メモリー カラノ ヨミコミ。
80 PRINT " ";HEX$(DT);
90 NEXT
RUN
 0 1 2 3 4 5 6 7 8 9 A B C
 D E F 10 11 12 13 14 15 16 17
 18 19 1A 1B 1C 1D 1E 1F
```

メモリー内のC000～C01Fの各番地に0～31の数値データを書き込み、実際に書き込まれたかどうか調べるため読み込んで、16進で表示させています。

2.2.33 OUT

機能

出力ポートに1バイトのデータを送ります。(⇒POKE@)

書式

```
OUT ポートアドレス, 出力データ
```

ポートアドレス…出力するポートの番号—32768～65535

出力データ………0～255の整数

省略形

OU.

解説

出力データをポートアドレスで指定したポートに出力します。

テキスト画面とその属性ポートへのアクセスの仕方については、巻末の「テキスト画面とその属性ポートへのアクセス方法」を参照してください。

画面の左上隅（&H3000番地）から256文字表示し、そのおのおのに点滅、反転、色指定の各属性をつけるプログラムを下に示します。

サンプルプログラム

```
10 SCREEN0,0,0:WIDTH 40
20 FOR I=0 TO 255
30  OUT &H3000+I,I
35  OUT &H2000+I,(I MOD 32)
40 NEXT
50 LOCATE 0,23
```

2.2.34 RANDOMIZE

機　　能　RND関数で発生させる乱数の系列を設定します。

書　　式

```
RANDOMIZE [式]
```

式…乱数系列を発生するための初期値。 −32768～32767。

省 略 形　RA.

解　　説

RND関数はプログラムを実行するたびに同じ系列の乱数を発生しますが、このステートメントを与えることによって、乱数の系列を変更します。

式は-32768～32767までの数値になるように指定しますが、省略した場合は、BASICがランダムに乱数系列を設定します。

サンプルプログラム

```
LIST
10 PRINT RND(1)
20 PRINT RND
30 RANDOMIZE 10
40 PRINT RND
50 RANDOMIZE 10
60 PRINT RND
OK
RUN
 .81971014
 .405249
 .50813067
 .50813067
OK
```

(注)
この値は乱数表すので
サンプル通りでるとは限りません。

2.3 ファイル処理ステートメント

2.3.1 MAXFILES

機能　同時にOPENできるファイル番号の最大値を指定します。

書式

```
MAXFILES n
```

n…ファイル番号の最大値 0 ～15。

省略形　MA.

解説　同時にファイル番号の最大値nまでのファイルをOPENできるように設定します。このとき、変数エリアはクリアされます。

BASIC起動時にファイル番号の最大値は1に指定されます。

サンプルプログラム

```
MAXFILES 3
OK
```

2.3.2 OPEN

機能　ファイルを開きプログラムで使用できるようにします。

書式

```
OPEN "{O|I}",[#]ファイル番号,"ファイルディスクリプタ：ファイル名"
```

ファイル番号…1からMAXFILESで設定した数まで。

省略形　OPE.

解説　ファイルディスクリプタで指定したファイル名をつけて、ファイル番号で指定した番号で設定し、Oのときはアウトプットモード、Iのときはインプットモードにします。

以後このファイルは、ファイル番号を使ってアクセスします。

O…シーケンシャルファイルのアウトプットオープン

I…　　〃　　インプットオープン

サンプルプログラム

```
10 REM SAMPLE PROGRAM
20 OPTION SCREEN 2          'グ゛ラフィックメモリヲ ファイルニ ツカエルヨウニ スル
30 INIT"MEM:"               'ショキカ スル
40 OPEN "O",#1,"MEM:TEST"   'カキコミファイルヲ ヒラク
50 FOR I=0 TO 255
60 PRINT #1,I;              'ファイルニ カキコム
70 NEXT
80 CLOSE                    'ファイルヲ トシ゛ル
90 OPEN "I",#1,"MEM:TEST"   'ヨミコミファイルヲ ヒラク
100 IF EOF(1)=-1 THEN 140   'テ゛ータカ゛オワッタカ ト゛ウカ シラヘ゛ル
110 INPUT#1,A               'ファイルカラ ヨミコム
120 PRINTA;
130 GOTO 100
140 CLOSE                   'ファイルヲ トシ゛ル
150 END
```

2.3.3 CLOSE

機　　能　ファイルを閉じます。

書　　式

```
CLOSE[[#]ファイル番号1,[#]ファイル番号2,…]
```

ファイル番号…OPENで指定したファイル番号。

省 略 形　CLO.

解　　説　OPENによって開かれたファイルを閉じます。CLOSEで閉じたファイルはOPENで再指定しないかぎり、使用できません。

ファイル番号を省略するとすべてのファイルが一斉に閉じられます。

なお、NEW、RUN、LOADのときもすべてのファイルは閉じられます。

OPEN文参照。

サンプルプログラム

```
CLOSE
CLOSE #1,#3
```

2.3.4 PRINT#

機　　能　シーケンシャルファイルにデータを記録します。

書　　式

```
PRINT#ファイル番号[,x1,x2,…]
```

ファイル番号…OPENで指定したファイル番号

$x_1,x_2,$……………出力する式または文字列

省 略 形　?#　またはP.#

解　　説　$x_1,x_2,$…で指定した式または文字列をファイル番号で指定したファイルへ出力して記録します。このステートメントを実行するためには、OPEN(OPEN文参照)ですでにファイルを開いておかねばなりません。なお、一連の処理が終了したら必ずCLOSE(CLOSE文参照) してください。

PRINT#0は、CSIZE (CSIZE文参照) と対にして使い、式の値、文字列、変数・定数の値に加えてコントロールコード（00～1Fに対応する）も画面に表示することができます。

なお、PRINT#0は、OPENしないでも使えます。

OPEN文参照

サンプルプログラム

```
PRINT#1,A,B,C
PRINT#0,X;"X"
```

2.3.5 WRITE＃

機　　能　データをシーケンシャルファイルに記録します。

書　　式

WRITE＃ファイル番号[,x_1,x_2,…]

　　ファイル番号…OPENで指定したファイル番号
　　x_1,x_2,…………出力する式または文字列

省 略 形　WR.＃

解　　説
　ファイル番号で指定したシーケンシャルファイルに、x_1,x_2,…で指定した式または文字列を記録します。WRITE＃はPRINT＃と異なり、データの区切りにカンマ（,）を入れ、文字列データのときはダブルコート（"）をつけて、詰めて出力するので。ファイルの使用領域の節約になります。

サンプルプログラム

```
WRITE#1,A,B,C
WRITE#0,X;"X"
```

2.3.6 INPUT＃

機　　能　シーケンシャルファイルからデータを変数に読み込みます。

書　　式

INPUT＃ファイル番号,x_1[,x_2,…]

　　ファイル番号…OPENで指定したファイル番号
　　x_1,x_2,…………入力したデータを入れる変数

省 略 形　I.＃

解　　説
　ファイル番号で指定したファイルからシーケンシャル（記録されている順）に読み込んで変数x_1，x_2,…へ入れます。このステートメントを実行するためにはOPEN(OPEN文参照)で､すでにファイルを開いておく必要があります。なお、一連の処理が終了したら必ずCLOSE(CLOSE文参照）してください。

サンプルプログラム

```
INPUT#1,A,B,C
```

2.3.7 LINPUT＃・LINE INPUT＃

機　能

シーケンシャルファイルから1行分（255文字まで）のデータを読み込みます。

書　式

```
LINPUT＃ファイル番号，文字変数
LINE INPUT＃ファイル番号，文字変数
```

ファイル番号…OPENで指定したファイル番号
文字変数………入力したデータを入れる文字型変数

省略形

LI.＃　LINEI.＃

解　説

ファイル番号で指定したシーケンシャルファイルから255文字までのデータを読み込んで文字変数に入れます。

サンプルプログラム

```
LINPUT#1,A$
LINE INPUT#1,A$
```

2.3.8 DEVI$

機　能

指定された入力デバイスから1レコードのデータを読み込みます。

書　式

```
DEVI$ "ファイルディスクリプタ：" ,レコード番号，文字変数，文字変数
```

ファイルディスクリプタ…CAS：　カセットテープ
（入力デバイス名）　　　MEM：　グラフィックメモリー
EMM 0：外部メモリー 0
　∫　　　∫
EMM 9：外部メモリー 9
レコード番号………………各入力デバイスの先頭からのレコード番号
1レコード＝256バイト

省略形

なし

解　説

ファイルディスクリプタで指定した入力デバイス内のレコード番号で指定したレコードから、256バイト分の文字列データを読み込み128バイトずつ2つの文字変数に入れます。

サンプルプログラム

```
DEVI$"EMM0:",1,A$,B$
```

2.3.9 DEVO$

機能

1レコード分の文字列データを指定された出力デバイスに記録します。

書式

```
DEVO$ "ファイルディスクリプタ：", レコード番号, 文字式, 文字式
```

ファイルディスクリプタ…CAS： カセットテープ
MEM： グラフィックメモリー
EMM0：外部メモリー0
〜 〜
EMM9：外部メモリー9
レコード番号………………データを記録するレコードの番号。

省略形

DEVO.

解説

ファイルディスクリプタで指定した出力デバイス内の、レコード番号で指定したレコードに、文字式の文字列データを256文字分記録します。このとき1つの文字式には128バイトの文字が入っている必要があります。

サンプルプログラム

```
DEVI$"EMM0:",1,A$,B$
```

2.3.10 INIT

機能

外部デバイスの初期化を行います。

書式

```
INIT["ファイルディスクリプタ："]
```

ファイルディスクリプタ…外部デバイスを指定する（ファイルディスクリプタ参照)。

省略形

なし。

解説

指定した外部デバイスの初期化（イニシャライズ）を行います。

ファイルディスクリプタ	処理内容
CRT： SCR：	COLOR7,0：CGEN：CFLASH：CREV：CSIZE ：PALET：PRW：WINDOW：CONSOLE：SCREEN 0,0,0
KEY： LPT：	Bad file modeエラーが出ます。
CAS：	カセットの巻き戻しと先頭からの消去をします。
EMM0： 〜 EMM9： MEM：	フォーマットの管理テーブルを初期化します。

"ファイルディスクリプタ" を省略すると、スクリーン（CRT：,SCR：）の初期化が行われます。

サンプルプログラム

```
INIT"CRT:"
Are you sure ? (y or n)Y
Ok
```

ダイレクトモードでINITを実行すると、「Are you sure?(YorN)」と表示されますので、実行するときは[Y]キーを押し、実行を中止するときはそれ以外のキーを押してください。

プログラム中でINITを実行しても「Are you sure?(YorN)」と聞いて来ないで、直接実行してしまいますので注意してください。

2.4 エラー処理ステートメント

2.4.1 ON ERROR GOTO

機　能

エラーが発生したとき指定行番号からのエラー処理ルーチンに制御を移します。

書　式

ON ERROR GOTO { 行番号 / "ラベル名" }

行　番　号 …エラーが発生したとき実行させるエラー処理ルーチンの開始行番号。
"ラベル名"…　　　　　　　　〃　　　　　　　　ラベル名。

省 略 形

O. ERR. G.

解　説

このステートメントを前もって実行していると、エラーが発生したときに、行番号で指定した行からのエラー処理ルーチン（ユーザーが設定）へ制御を移し、処理ルーチンの出口のRESUME(RESUME文参照）までの実行を行います。

これによって、ほとんどのエラーに対し、BASICレベルで対策可能になります。なお、ON ERROR GOTO 0とすると、このステートメントが止まり、通常のエラーが発生します。エラー処理ルーチンの中でエラーが起きた場合は、エラーを出力してコマンド待ちにもどります。

またエラー処理ルーチンを実行中、プログラムの最後に達した場合は、NO RESUMEエラーが発生します。END文で終った場合は、エラーが発生しないで、コマンド待ちにもどります。
普通このステートメントは、プログラムの先頭で実行しておきます。

サンプルプログラム

```
LIST
10 ON ERROR GOTO 1000
20 INPUT"A,B=",A,B
30 PRINTA/B
40 GOTO 20
1000 IF ERR<>11 THEN ON ERROR GOTO 0
1010 PRINT"ｹｲｻﾝ ﾃﾞｷﾏｾﾝ !!!";CHR$(7)
1020 RESUME NEXT
Ok.
RUN
A,B=10,2
 5
A,B=10,0
ｹｲｻﾝ ﾃﾞｷﾏｾﾝ !!!
A,B=10,3
 3.3333333
A,B=
Break in 20
Ok.
```

← SHIFT BREAK を押す。

2.4.2 RESUME

機　能　エラールーチン終了後、プログラムの実行を再開します。

書　式

```
RESUME ⎛⎧ 行番号     ⎫⎞
       ⎜⎨ "ラベル名" ⎬⎟
       ⎝⎩ NEXT       ⎭⎠
```

行番号、"ラベル名"…エラー処理ルーチン終了後復帰する行番号・ラベル

省略形　RESU.

解　説　ON ERROR GOTOで飛び込んだエラー処理ルーチンから戻る場合は、必ずRESUMEによらなければなりません。

RESUMEのみならば、エラーの発生した命令から、

RESUME NEXTならば、エラーの発生した次の命令から、

RESUME 行番号
　〃　"ラベル名"　}ならば、その行番号またはラベルから、

実行を再開します。

もし、RESUMEではなく、GOTOなどで、もとのプログラムに実行を移した場合は、2回目のエラーが発生したときに、エラー処理ができなくなります。

サンプルプログラム

```
RESUME "ﾓﾄﾉ ﾙｰﾁﾝ"
```

2.4.3 ERROR

機　能　エラーを故意に発生させます。

書　式

```
ERROR　エラーコード
```

エラーコード…BASICで定義されているエラーコード

解　説　エラーコードで指定したエラーが発生し、ERR関数(ERR参照)にはエラーコードが、ERL変数(ERL参照)にはそのステートメントの行番号が入ります。これによって、エラー処理ルーチンが正常に作動するか確めることが可能です。エラー処理ルーチンのデバッグが終了したら、プログラム中より取りのぞいて下さい。

サンプルプログラム

```
ERROR 3
RETURN without GOSUB
OK.
ERROR 15
String too long
OK.
```

2.5 画面制御ステートメント

2.5.1 WIDTH

機能

画面サイズを設定します。

書式

WIDTH　一行当たりのカラム数

一行当たりのカラム数…40、80
（電源投入時は40カラム25ラインに設定されます。）

省略形

WI.

解説

画面のサイズを設定するときに使います。
一行当たりのカラム数を40に指定すると、画面のサイズは40カラム25ラインに設定され、80に指定すると、80カラム25ラインに設定されます。
このステートメントを実行すると、WINDOW（WINDOW文参照）とCONSOLE（CONSOLE文参照）を解除し、CLS4（CLS文参照）と同じ処理をします。
なお、一行当たりのカラム数に0～39を指定すると、40のときと同じ画面サイズに、41～255を指定すると、80のときと同じサイズに設定されます。

サンプルプログラム

```
WIDTH 40
WIDTH 80
```

2.5.2 SCREEN・GRAPH

機　　能　グラフィック画面の使用モードを設定します。

書　　式

```
SCREEN [[出力ページ][,[入力ページ][, グラフィックモード]]]
```

出力ページ……………0，1（0のときページ0、1のときページ1を指定します。）
入力ページ……………0，1（　　　　〃　　　　）
グラフィックモード…0，1，2，3

省 略 形　SC.
GR.

解　　説　SCREENとGRAPHは同じ命令で、出力・入力のページ指定とグラフィックモードを指定します。
出力・入力のページとはWIDTH40の時のみ有効で、2ページのうち、どちらを表示してどちらに書き込むか指定するものです。書き込むページというのは、画面制御命令やPRINT、INPUT命令等が実行されるページであり、出力ページとは、実際画面に出されているページのことをいいます。
また、グラフィックモードとはG1、G2、G3の3枚のグラフィック画面の使用モードで、この命令実行以後のグラフィック命令は次に示す画面に対してのみ有効となります。
グラフィックモード0…指定のパレット番号でドットをセット・リセットできます。
　　〃　　　　　1…グラフィック1のみドットをセット・リセットできます。
　　〃　　　　　2…グラフィック2のみドットをセット・リセットできます。
　　〃　　　　　3…グラフィック3のみドットをセット・リセットできます。
この命令はPALETおよびPRWも制御しており、すべてのパラメーターを省略した場合は、グラフィックが見えなくなり(LIST時も同じ)、パラメーターを指定した時には、省略した場合やLIST等で見えなくなる前のPALET、PRWにもどします。カーソルが見えなくなってこまった時は、CTRL + D を押して下さい。

サンプル プログラム

```
SCREEN 0,0,0
SCREEN 0,1,0
SCREEN 0,1,1
```

2.5.3 CONSOLE

機　能　画面内での文字の表示エリアを設定します。

書　式

```
CONSOLE  Ys, Yℓ[, Xs, Xℓ]
```

Y_s……垂直方向の表示開始カラム　0～24
Y_ℓ……垂直方向の表示カラム数　1～25(正確には1～25-Y_s)
X_s……水平方向の表示開始カラム
　WIDTH 40のとき　0～39
　WIDTH 80のとき　0～79
X_ℓ……水平方向の表示カラム数
　WIDTH 40のとき　1～40(正確には1～40-X_s)
　WIDTH 80のとき　1～80(正確には1～80-X_s)

省略形　CONS.

解　説　テキスト画面に対して、文字の表示エリアを設定します。
このステートメントの実行後は、指定した長方形のエリア内だけに文字を表示することができ、画面のクリアやスクロールもこのエリア内で行われます。設定後カーソルは(X_s、Y_s)の位置へ移動します。
CONSOLEの後を省略すると、最大文字数表示する画面に戻ります。

サンプルプログラム

```
LIST
10 CLS
20 CONSOLE 0,10,0,40
30 FOR X=1 TO 500
40  PRINT X;
50 NEXT
60 CONSOLE 0,25,0,20
70 FOR X=1 TO 500
80  PRINT X;
90 NEXT
Ok
```

このプログラムを実行すればCONSOLEの設定の違いが分かります。

2.5.4 LOCATE・CURSOR

機　　能　カーソルを指定位置へ移動します。

書　　式

```
LOCATE x, y
```

```
CURSOR x, y
```

x, y……CONSOLE命令で指定された表示エリア内の値。
　x……水平方向の表示カラム　WIDTH40のとき　0〜39
　　　　　　　　　　　　　　WIDTH80のとき　0〜79
　y……垂直方向の表示カラム　WIDTHに関係なく　0〜24

省 略 形　LOC.
CU.

解　　説　LOCATEとCURSORは同じ命令で、テキスト画面のカーソル位置を指定します。範囲はCONSOLEで指定された表示エリア内で、それを外れた場合はエラーになります。

サンプルプログラム

```
LIST
10 FOR I=0 TO 39
20 FOR J=0 TO 24
30  LOCATE I,J:PRINT"*";
40 NEXT
50 NEXT
Ok.
```

WIDTH 40

WIDTH 80

2.5.5 COLOR

機　　能　画面の色を設定します。

書　　式

```
COLOR [表示色] [,背景色]
```

表示色…文字の表示色を下のような数字で指定します。

0 黒
1 青
2 赤
3 マゼンタ
4 緑
5 シアン
6 黄
7 白

これをカラーコードといい、省略すると表示色は変化しません。

背景色…表示色と同様にカラーコードで指定し、省略すると背景色は変化しません。

省 略 形　COL.

解　　説　文字の色と背景の色を指定する命令です。

電源投入時は、自動的にCOLOR 7, 0にセットされています。グラフィック画面において、パレットコード(PALET文参照)を省略すると、COLOR文の表示色のコードの値が、パレットコードの値として使用されます。

なお、表示色と背景色はCTRLキーと数字のキーを同時に押しても設定することができます。(「コントロールコード」参照)。

サンプルプログラム

```
COLOR 2

COLOR 6

COLOR 3,0

COLOR 4,0
```

2.5.6 CREV

機　　能　文字の表示モードの切り換えをします。

書　　式

```
CREV [文字表示モード]
```

文字表示モード…0, 1

省略及び0のとき　ノーマルモード(標準モード)

1のとき　リバースモード(反転モード)

省 略 形　CR.

解　　説　文字の表示モードを反転モードにしたり標準モードにしたりするためのステートメントです。

このステートメントの実行以降、文字は指定された表示モードで画面に表示されます。

ノーマルモード　A, B, C, …, 1, 2, 3, …, _ (ブランク)

反 転 モ ー ド　A, B, C, …, 1, 2, 3, …, ■

サンプルプログラム

CFLASH参照。

2.5.7 CFLASH

機　能　文字の表示モードの切り換えをします。

書　式

```
CFLASH [文字表示モード]
```

文字表示モード…0，1
省略及び0のとき　ノーマルモード　（標準モード）
　　　1のとき　フラッシングモード(点滅モード)

省略形　CF.

解　説
文字の表示モードをフラッシングモードにしたり、ノーマルモードにしたりするためのステートメントです。
このステートメントの実行以降、文字は指定された表示モードで画面に表示されます。ノーマルモードはCREV (CREV文参照)のと同様です。フラッシングモードとは、CREVのノーマルモードと反転モードとを交互に繰返すモードです。このとき、切り変わりの周期は、画面上のカーソルの点滅の周期と同じになります。

サンプルプログラム

```
CFRASH 1
OK          ←┐
CFRASH 0    ←┘── 画面上では点滅しています。
OK
```

```
10 REM SAMPLE PROGRAM
20 INIT:WIDTH 40
30 LOCATE 5,5
40 CREV 0:CFLASH 0
50 PRINT"NORMAL           CHARACTER";
60 LOCATE 5,7
70 CREV 1:CFLASH 0
80 PRINT"REVERSE          CHARACTER";
90 LOCATE 5,9
100 CREV 0:CFLASH 1
110 PRINT"FLASH            CHARACTER";
120 LOCATE 5,11
130 CREV 1:CFLASH 1
140 PRINT"REVERSE & FLASH CHARACTER";
150 CREV:CFLASH
160 END
```

2.5.8 CGEN

機　　能　文字の表示モードの切り換えをします。

書　　式

CGEN［文字表示モード］

文字表示モード…0，1
省略及び0のとき　ノーマルモード　　(ROMCGモード)
　　　　1のとき　ユーザー文字モード(RAMCGモード)

省 略 形　CG.

解　　説　文字の表示モードをROMCG(ROMキャラクタジェネレータ)モードにしたり、RAMCG(RAMキャラクタジェネレータ)モードにしたりするためのステートメントです。
RAMCGモードにすると、それ以後表示される文字はすべて、ユーザー定義のキャラクタとなります。
BASIC起動時は何も定義されていないので、DEFCHR$命令で定義してから使用してください。

サンプルプログラム

```
CGEN 1
CGEN 0
```

2.5.9 CSIZE

機　　能　文字の大きさを変えます。

書　　式

CSIZE［n］

n………0，1，2，3
省略及び0　ノーマル文字
　　　　1　垂直2倍文字
　　　　2　水平2倍文字
　　　　3　垂直2倍・水平2倍文字

省 略 形　CS.

解　　説　PRINT＃0命令で表示する文字の大きさを指定します。ノーマル以外を指定した場合、PRINT＃0命令を実行する時、次の注意が必要です。
垂直2倍文字…その二行中に、ノーマル及び水平2倍文字があってはなりません。
水平2倍文字…必ず偶数のX座標から出力すること。一行中にノーマル文字があってもかまいません。
垂直2倍・水平2倍文字…その二行中にノーマル及び水平2倍文字があってはなりません。垂直2倍文字はあってもかまいません。また、必ず偶数のX座標から出力してください。

サンプルプログラム

```
100 A$="ABCDEFG":CLS
110 CSIZE 0
120 LOCATE 0,0:PRINT#0 A$
130 CSIZE 1
140 LOCATE 0,1:RRINT#0 A$
150 CSIZE 2
160 LOCATE 0,3:PRINT#0 A$
170 CSIZE 3
180 LOCATE 0,4:PRINT#0 A$
190 CSIZE 0
200 END
```

2.5.10 DEF CHR$

機　　能　ユーザー定義のキャラクタジェネレータにキャラクタパターンを定義します。

書　　式

DEF　CHR$(n)＝文字式

n………0～255 キャラクタコード。

文字式…24文字のパターンデータ。

省 略 形　DEFCHR.

解　　説　ユーザーが定義できるキャラクタは、通常のキャラクタ(文字)と同様に横方向に8ドット、縦方向に8ドットの長さをもつタイル型のドットパターンです。このキャラクタを定義するためには、8×3＝24文字の文字列が必要で、1文字で横8ドット、縦1ドットのパターンを指定し、8文字で縦横8ドットの長さのキャラクタを指定します。すなわち、1文字に対応するキャラクタコードのビットパターンが、横8ドットのドットパターンを表します。

文字列の最初の8文字は青、次の8文字は赤、最後の8文字は緑のキャラクタパターンを示します。

ユーザーの定義できるキャラクタの数は256個までで、キャラクタコードnに対応しています。表示するには、前もってCGEN 1 (CGEN文参照)を実行し、PRINT文を使います。

サンプル プログラム　キャラクタとして斜線を定義して、8つのカラーで表示してみましょう。

HEXCHR$ ("01 02 04 08 10 20 40 80")のパターンは下の様になります。

ドットパターン

01＝$(00000001)_2$

02＝$(00000010)_2$

04＝$(00000100)_2$

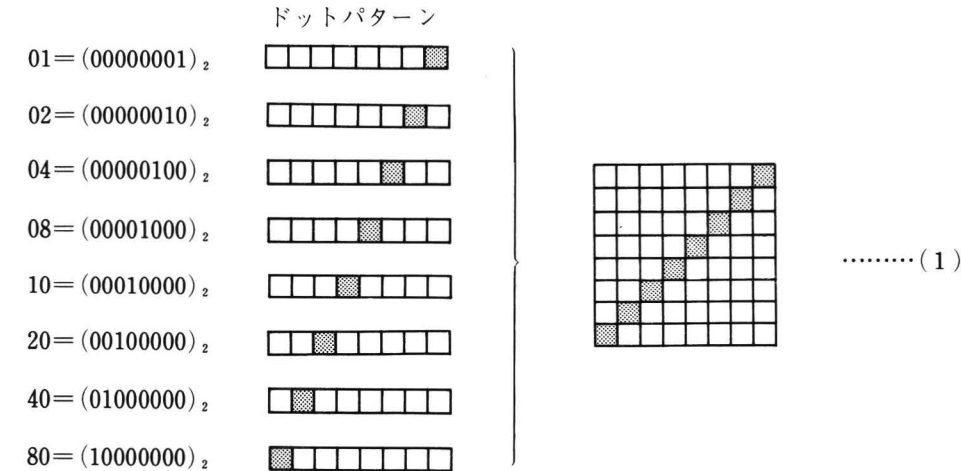
08＝$(00001000)_2$

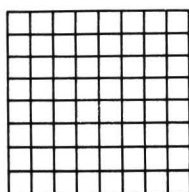
10＝$(00010000)_2$

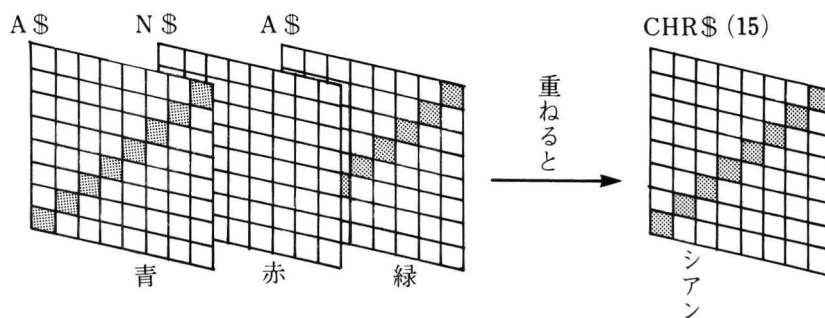
20＝$(00100000)_2$

40＝$(01000000)_2$

80＝$(10000000)_2$

………(1)

```
10 SCREEN 0,0,0:CGEN 1
20 A$=HEXCHR$("0102040810204080")
30 N$=HEXCHR$("0000000000000000")
40 '
50 '                  アオ  アカ  ミドリ
60 '                   |    |    |
69 DEF CHR$(10)=N$+N$+N$
70 DEF CHR$(11)=A$+N$+N$
80 DEF CHR$(12)=N$+A$+N$
90 DEF CHR$(13)=A$+A$+N$
95 DEF CHR$(14)=N$+N$+A$
96 DEF CHR$(15)=A$+N$+A$
97 DEF CHR$(16)=N$+A$+A$
98 DEF CHR$(17)=A$+A$+A$
110 '
130 FOR I=10 TO 17
140 PRINT#0,CHR$(I);
150 NEXT
200 CGEN 0:END
```

HEXCHR＄("0000000000000000")は、全部ドットが消えている状態で、下の様になります。

(1)のパターンをA＄、(2)のパターンをN＄として、これらを組み合せてCHR＄(10)〜CHR＄(17)の8色の斜線パターンを定義します。

たとえば、CHR＄(15)はA＄＋N＄＋A＄として

10〜17はコントロールコードなので、PRINT＃0を使って表示します。

2.5.11 WINDOW

機　　能　グラフィック画面の表示エリアを設定します。

書　　式

```
WINDOW (Xs,Ys)−(Xe,Ye) [,(X1,Y1),(X2,Y2)]
```

X_s, Y_s……始点座標。

X_s…水平座標　WIDTH40のとき0〜319
WIDTH80のとき0〜639

Y_s…垂直座標　0〜199

X_e, Y_e……終点座標。

X_e…X_sに同じ

Y_e…Y_sに同じ

X_1, Y_1……始点の論理座標。-1.7014118E＋38〜1.7014118E＋38。

X_2, Y_2……終点の論理座標。-1.7014118E＋38〜1.7014118E＋38。

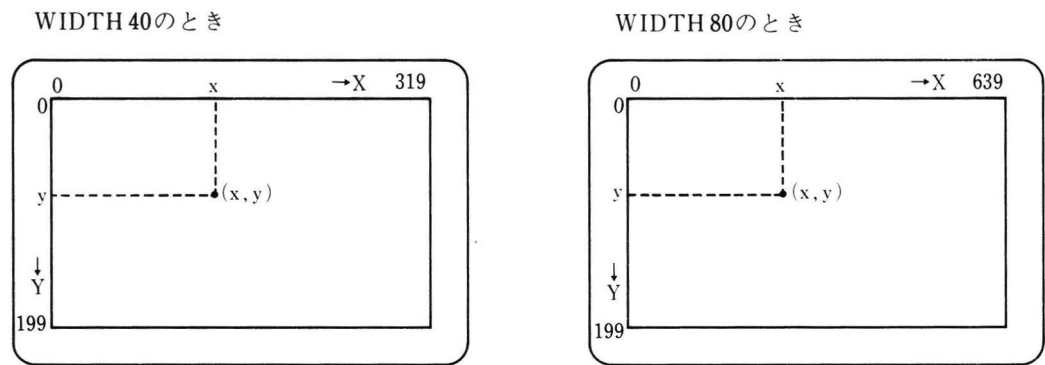
省 略 形　WIN.

用語の説明

〈グラフィック画面の座標系〉

グラフィック画面の分解能は、WIDTH 40のとき320×200、WIDTH 80のとき640×200で、各ドットの位置は下の図のような座標を使って表します。

WIDTH 40のとき

WIDTH 80のとき

〈絶対座標系と論理座標系〉

グラフィック画面の座標系には、絶対座標系と論理座標系の2種類があり、グラフィック画面に対してのみ意味をもちます。

○絶対座標系

上の〈グラフィック画面の座標系〉で説明した座標に一致するもので、画面上の1ドットが座標単位の1に対応します。WIDTH40とWIDTH80とでは水平方向(X軸方向)の大きさが、0〜319、0〜639と異なります。

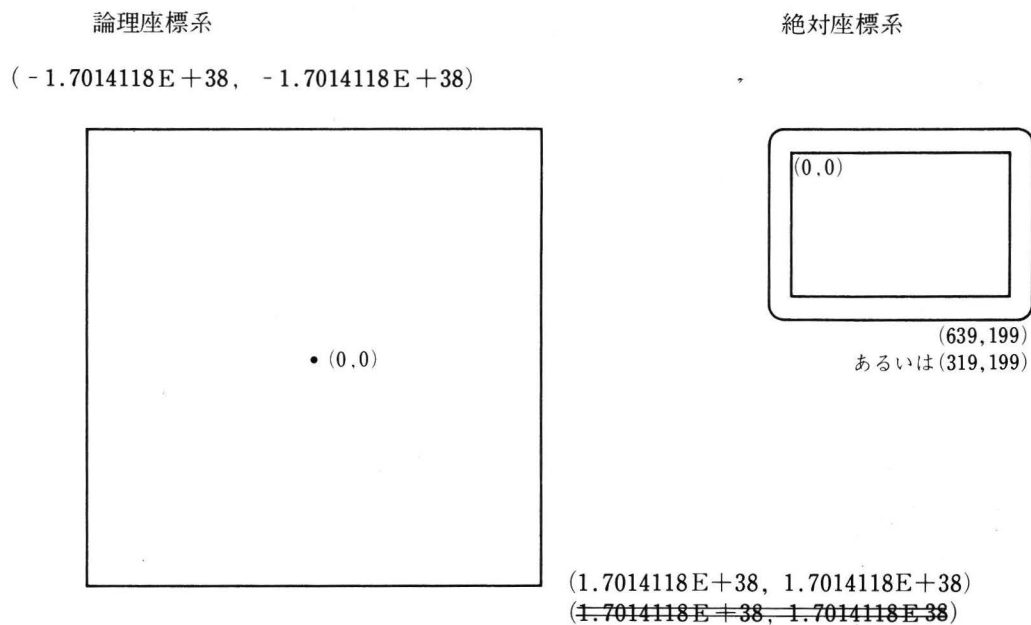
○論理座標系

WINDOW命令によってユーザーが指定した座標系で、画面の垂直方向・水平方向とも -1.7014118E＋38〜1.7014118E＋38までの単精度の範囲で指定できます。後に詳述するPSET、PRESET、LINE、CIRCLE、POLY、PAINTのグラフィックステートメント、およびPOINT関数はこの論理座標系に対して機能します。

下の図は、この座標系と絶対座標系の概念図です。

論理座標系　　　　　　　　　　　　　　　　　　　　　　絶対座標系

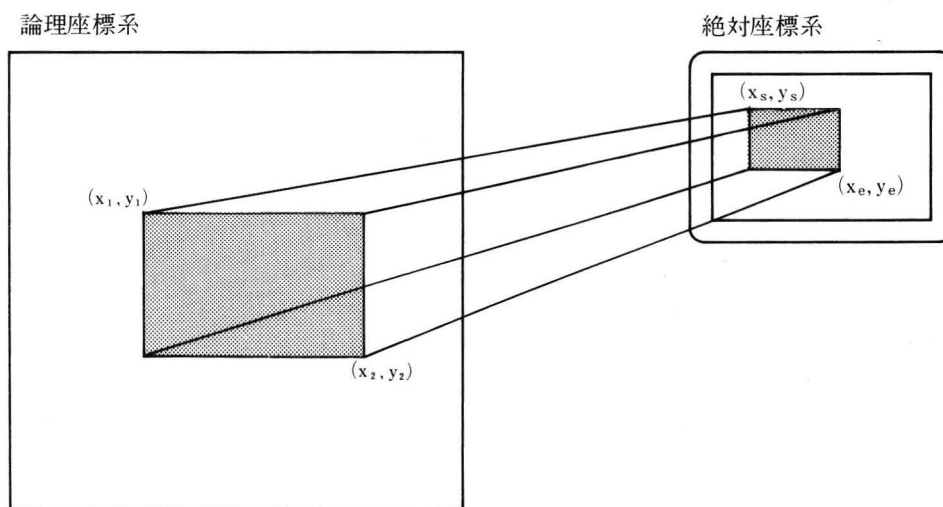
(－1.7014118E＋38，－1.7014118E＋38)

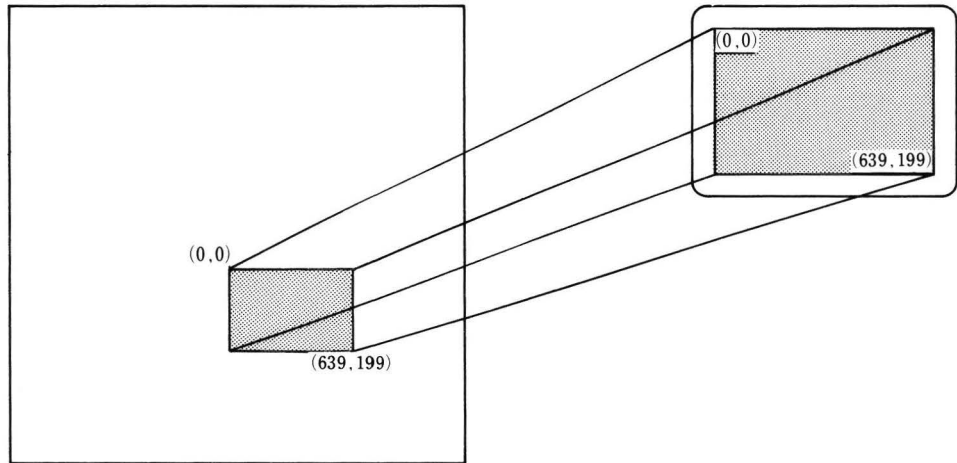
(1.7014118E＋38，1.7014118E＋38)
(~~1.7014118E＋38，1.7014118E38~~)

★広大な平面を想像してください。

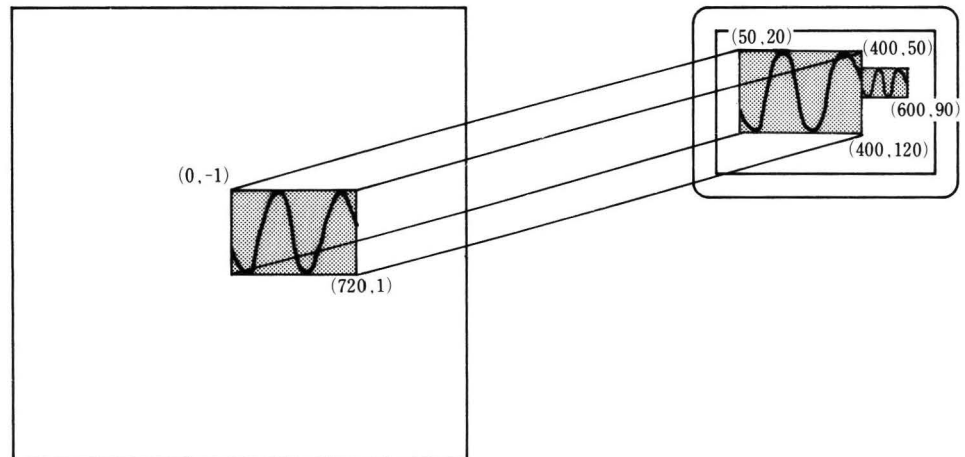
解　説

ユーザーはWINDOW命令を使うことによって、広大な論理座標平面の中から任意の領域を指定して、画面(絶対座標平面)上の任意の領域にそれを表示させることができます。たとえば、論理座標系の(x_1, y_1)から(x_2, y_2)を対角線とする長方形の領域を、画面上の(x_s, y_s)から(x_e, y_e)を対角線とする長方形の領域に表示させるには、

　WINDOW (x_s, y_s)-(x_e, y_e), (x_1, y_1)-(x_2, y_2)と指定します。

下にその概念図を示します。

このように、論理座標系(単精度)の範囲内ならば、マクロ・ミクロのどんな領域でも、画面に表示することができます。

論理座標の指定を省略すると、絶対座標系のサイズと同じ領域が自動的に指定されます。

サンプル
プログラム

0°～720°までのサインカーブを描いてみます。論理座標で水平方向に0～720、垂直方向に-1～1の枠を設定し、絶対座標の(50, 20)と(400, 120)を対角線とする長方形の領域に、その後(400,50)と(600, 90)を対角線とする長方形の領域にサインカーブを描きます。

```
10 SCREEN0,0,0:WIDTH 80
20 WENDOW(50,20)-(400,120),(0,-1)-(720,1)
30 S=2:GOSUB 100
40 WENDOW(400,50)-(600,90),(0,-1)-(720,1)
50 S=4:GOSUB 100
60 LOCATE 0,20:END
100 FOR X=0 TO 720 STEP S
110 Y=SIN(RAD(X))
120 PSET(X,Y)
130 NEXT
140 RETERN
```

2.5.12 CLS

機　　能　画面をクリアします。

書　　式

```
CLSn
CLS
```

n＝0，1，2，3，4

省 略 形　CL.

解　　説　画面表示を消して、背景色のみの画面にします。

nの値によって、グラフィック1、グラフィック2、グラフィック3のそれぞれをアクセスします。

nの値

0　グラフィック1、グラフィック2、グラフィック3を同時にクリアします。
1　グラフィック1をクリアします。
2　グラフィック2をクリアします。
3　グラフィック3をクリアします。
4　グラフィック1、グラフィック2、グラフィック3とテキスト(文字)を同時にクリアします。
省略　テキスト画面をクリアします。CONSOLE(CONSOLE文参照)で画面が指定されているときには、指定された範囲のテキスト画面をクリアします。

グラフィック画面をクリアする場合、WINDOW(WINDOW文参照)で画面が指定されているときには、指定された絶対座標(WINDOW文参照)の範囲の画面をクリアします。

サンプル プログラム

```
CLS        画面左上隅に表示
OK.      ← ~~画面左右隅に表示~~
CLS 0
OK.
CLS 4
OK.      ←       〃
```

2.5.13 PALET

機　能　パレットのカラーを設定します。

書　式

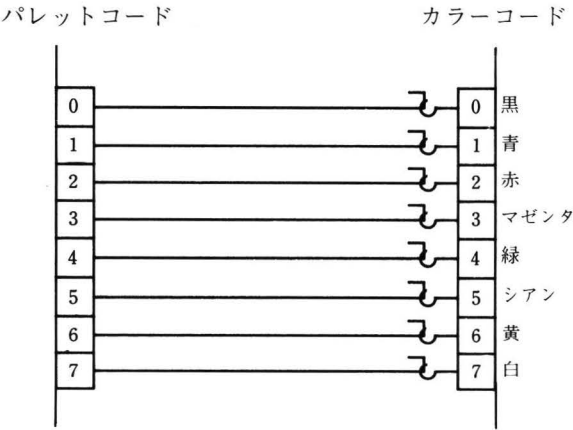
```
PALET [パレットコード, カラーコード]
```

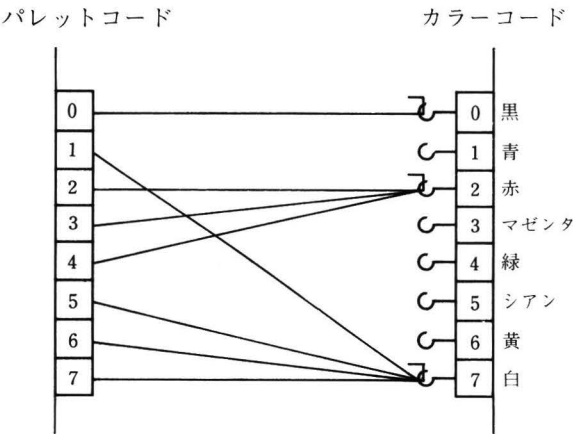
パレットコード…0～7
カラーコード……0～7 (COLOR文参照)

省略形　PAL.

解　説　パレットコードで指定されるパレットに、カラーコードで指定されるカラーを設定します。

SHARP HuBASICは、0～7のパレットコードで表される8つのパレットをもっているので、ユーザーはこのパレットに任意のカラーコードを設定して、カラーをダイナミックに使用することができます。

下の図は、パレットの概念を表すもので、パレットコードとカラーコードの間はゴムひもでつながっていて、パレットコード側は固定され、カラーコード側にはフックがついています。

BASIC起動時には、上の図のように、パレットコード0がカラーコード0(黒)、パレットコード1がカラーコード1(青)……というようにフックがかかっていて、パレットコード＝カラーコードとして使用することができます。

また、PALET1, 7：PALET3, 2：PALET4, 2：PALET5, 7：PALET6, 7と設定すると、下のようにフックがかかり、パレットコード0が黒、パレットコード2、3、4が赤、パレットコード1、5、6、7が白の3色使用できることになります。

フックの入れかえは瞬時に行えますので、プログラム中でパレット1が白のとき、画面には白い線が描けますが、PALET1, 4となった瞬間、白い線を緑色に切り替えるというような融通性に豊んだプログラムを作ることが可能です。

サンプル
プログラム

```
10 SCREEN0,0,0:WIDTH 40
30 LINE(0,0)-(300,150),PSET,1
40 FOR C=1 TO 7
50 PALET 1,C
60 FOR I=0 TO 200:NEXT I
70 NEXT C:GOTO 40
```

画面を斜めに走る直線の色が、青、赤、マゼンタ、緑、シアン、黄色、白と変化します。
止めるときは CTRL + C を押すか、 SHIFT + BREAK を押してください。
線を消すときは、CLS 0 ⏎ と押してください。

2.5.14 PRW

機　能　テキスト画面のグラフィック画面に対する優先順位を設定します。

書　式

```
PRW [n]
```

n…0～255の整数(省略は0)

省略形　なし

解　説　テキスト画面とグラフィック画面のどちらを優先して表示するかを設定するステートメントです。nの値0～255(2進で&B00000000～&B11111111)の各ビットが、左の桁からパレットコードの7、6、5、4、3、2、1、0に対応していて、2進でビット1のところがグラフィックのパレット優先、ビット0のところがテキスト画面優先となります。
たとえば、nが100のとき2進表現では01100100なので、パレットコードが6、5、2のものの表示がテキスト画面より優先されます。パレットコードの指定がないうちは、パレットコードはカラーコードに一致するので、テキスト画面のキャラクタ(文字)が黄色、シアン、赤の陰に隠れることになります。

ビット	7	6	5	4	3	2	1	0
	白	黄	シアン	緑	マゼンタ	赤	青	黒

0のとき　テキスト優先
1のとき　グラフィック優先

サンプルプログラム

```
10 SCREEN0,0,0:WIDTH 40
15 PRINT"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
20 FOR I=0 TO 7
30 LINE(I*40,0)-(I*40+39,199),PSET,I,BF
40 NEXT
45 FOR T=0 TO 5000:NEXT
50 PRW &B11010101
60 FOR T=0 TO 5000:NEXT
70 PRW &B10101010
75 FOR T=0 TO 5000:NEXT
80 CLS4:END
```

このプログラムを実行すると、はじめキャラクタAが1行と、カラータイルが表示され、しばらく経つとキャラクタAが黄色、緑、赤、黒の陰に隠れて見えなくなり、続いて、シアン、マゼンタ、青の陰に隠れて見えなくなります。
画面右端の白いタイルの部分はキャラクタAの保護色となっているため、Aは見えません。

2.5.15 CANVAS

機　　能　マルチ画面モード時の各グラフィック画面の色を指定します。

書　　式

```
CANVAS c1, c2, c3
```

c_1…グラフィック１の画面のカラーコード（COLOR文参照）
c_2…グラフィック２の画面のカラーコード
c_3…グラフィック３の画面のカラーコード

省 略 形　CAN.

解　　説　マルチ画面モード時のグラフィック画面１、２、３のそれぞれの色を指定します。グラフィック画面がかさなっている部分の表示色はLAYERで指定された優先順位により、画面１、２、３いずれかの色となります。
PALET命令やPRW命令と同時に使わないで下さい。

サンプルプログラム

```
CANVAS 1,2,3
CANVAS 7,2,4
```

2.5.16 LAYER

機　　能　テキスト画面とグラフィック画面の優先順位を指定します。

書　　式

```
LAYER  t, g1, g2, g3
```

t, g_1, g_2, g_3…1、2、3、4の4つの数が1対1に対応。

- t　テキスト画面の優先順位番号。
- g_1　グラフィック1の画面の優先順位番号。
- g_2　グラフィック2の画面の優先順位番号。
- g_3　グラフィック3の画面の優先順位番号。

省 略 形　LAY.

解　　説　テキスト画面(文字)とグラフィック画面3つの表示の優先順位を指定します。

t、g_1、g_2、g_3にはそれぞれ1番から4番の順位が入り、画面には順位の高い画面が優先的に表示されます。

PALETやPRWと同時には使わないでください。

サンプルプログラム

```
CLS0                                   ←グラフィック画面をクリアする。
OK
CANVAS 1,2,4                           ←画面1の色を青、画面2の色を赤、画面3の色
OK                                       を緑と決めます。
LAYER 1,2,3,4                          ←画面の優先順位を最優先のテキストから順にグ
OK                                       ラフィック画面1, 2, 3の順に設定します。
SCREEN 0,0,1                           ┐グラフィック画面1をアクセスするモードに設
OK                                     │定します。
LINE(0,60)-(200,160),PSET,1,BF        ┐これで長方形を書きます。CANVAS 1 (青)のた
OK                                     │めに青い長方形が書かれます。
SCREEN 0,0,2                           ┐グラフィック画面2をアクセスするモードに設
OK                                     │定します。
LINE(80,40)-(220,120),PSET,1,BF       ┐長方形を書きます。CANVASの2のために赤い
OK                                     │長方形が書かれます。
SCREEN 0,0,3                           ┐グラフィック画面3をアクセスするモードに設
OK                                     │定します。
LINE(180,0)-(319,199),PSET,1,BF       ┐長方形を書きます。CANVASの4のために緑の
OK                                     │長方形が書かれます。
LAYER 1,3,2,4                          ┐画面の優先順位を変更します。そのため今まで
OK                                     │青の長方形にかくれていた赤い長方形が前に出
                                       │てきます。
LAYER 1,4,3,2                          ┐画面の優先順位を変更します。テキスト、緑の
OK                                     │長方形、赤の長方形、青の長方形の順に長方形
                                       │がかさなります。
```

2.5.17 GET@

機能	グラフィック画面ドットデータを読み込みます。

書式

```
GET@ (x1, y1)-(x2, y2), 配列名 { , パレットコード }
                              { 省略            }
```

x_1, y_1 ……ドットデータ読み込み開始座標。｝(絶対座標)
x_2, y_2 ……　〃　終了座標。｝

配列名……数値型配列変数。(予めDIM文で用意する。)

省略形 なし。

解説 グラフィック画面上のドットデータを配列名で指定される配列変数に読み込みます。読み込む画面の領域は、開始座標(x_1, y_1)と終了座標(x_2, y_2)を対角とする長方形の枠内です。必要なカラー情報はパレットコードで指定しますが、この指定を省略するとテキストのデータを指定することになります。

読み込むのに必要な配列のバイト数は次の式で計算できます。

テキストの場合　(ABS(x_1 -x_2)+1)＊(ABS(y_1 -y_2)+1)＊2

グラフィックの場合　INT(((ABS(x_1 -x_2)+1)＊(ABS(y_1 - y_2)+1)+7)/8)＊m

mの値はパレットコードにより異なる…… 0の場合…m＝0

1,2,4　〃　…m＝1

3,5,6　〃　…m＝2

7　〃　…m=3

必要な配列の要素数は、必要なバイト数を、次の数で割ったものとなります。

整数型配列の場合……2

単精度　〃　……5

倍精度　〃　……8

必ず、予め必要バイト数を計算して、DIM文によって配列変数を用意してから実行するようにしてください。

サンプルプログラム PUT@参照。

2.5.18 PUT@

機 能 グラフィックドットデータをグラフィック画面に表示します。

書 式

PUT@ (x_1, y_1)-(x_2, y_2), 配列名 { , PSET, パレットコード / , PRESET, パレットコード / , XOR, パレットコード / , OR, パレットコード / , AND, パレットコード / , NOT, パレットコード / 省略(テキストの場合) }

x_1, y_1 ……ドット情報の読み込み開始座標。 } (絶対座標)
x_2, y_2 …… 〃 終了座標。

配列名……数値型配列変数。

(モード) PSET ……ビット1の点(dot)をつけ、ビット0の点を消す。
PRESET…ビット1の点を消す。
XOR ………ビット1の点を反転する。
OR…………ビット1の点をつける。
AND ………ビット0の点を消す。
NOT ………ビット1の点を消して、ビット0の点をつける。

省略形 なし。

解 説 GET@(GET@文参照)で配列変数に入れられたグラフィック画面のドットデータを、画面の指定した領域にグラフィックとして表示します。表示領域は開始座標(x_1, y_1)と終了座標(x_2, y_2)を対角とする長方形の枠内です。

モードについては、指定した領域内に既にあるドットのカラーデータについて、該当する配列変数中のドット情報と、カラービットごとの論理演算を行い、その結果を画面にセットします。テキスト情報の入った配列変数のときには、モード以降の設定はいりません。

GET@、PUT@の数値型配列変数名は( )を省略することができ、その場合、配列名(0)と同じになります。配列の添字を変えることによって、同一配列に2つ以上のパターンを定義することができます。

サンプルプログラム

```
CLS 0
10 INIT:CLS4
20 DIM A%(24)
30 POLY(5,5),5,7,144,0,720
40 POLY(5,5),5,4,144,90,810
50 GET@(0,0)-(10,10),A%,7
60 FOR I=0 TO 190 STEP 10
70 PUT@(I,I)-(I+10,I+10),A%,PSET,7
80 NEXT
```

2.5.19 POKE@

機　能　VRAM内の指定されたアドレスに1バイトのデータを書き込みます。(⇒OUT)

書　式

```
POKE@ アドレス, データ[, データ, データ…]
```

アドレス…VRAM内の指定アドレス。&H2000～&HFFFF
データ……メモリーに書き込む1バイトのデータ。0～255の整数

省略形　PO.@

解　説　VRAM内の指定したアドレスに、1バイトのデータを書き込みます。データを , (カンマ)で区切って続けて書くと、連続したアドレスに書き込むことができます。(付録「テキスト画面とその属性へのアクセス方法」参照)
画面のカーソル位置(&H3000番地)から256個の文字を表示し、その属性として、黒、青、赤、マゼンタ、緑、シアン、黄、白の色を指定するプログラムを下に示します。

サンプルプログラム

```
10 INIT:WIDTH 40
15 FOR I=0 TO 255
20 POKE@ &H3000+I,I
25 POKE@ &H2000+I,(I MOD 8)
30 NEXT
40 LOCATE 0,23
```

2.5.20 OPTION SCREEN

機　能　グラフィック用メモリーの使用目的の設定をします。

書　式

```
OPTION SCREEN n
```

n…1，2

省略形　OP. SC

解　説　グラフィック用メモリーの使用目的の設定に使います。
nの値
1．グラフィック用メモリーをグラフィック表示用に使います。
2．グラフィック用メモリーを外部デバイス用に使います。
OPTION SCREEN 2を行った後は、必ずINIT "MEM:" を実行してください。

サンプルプログラム

```
OPTION SCREEN 1
OK
OPTION SCREEN 2
OK
INIT"MEM:"
Are you sure ? (y or n)Y
OK
```

2.6 グラフィックステートメント

2.6.1 PSET

機　能　グラフィック画面に点（ドット）を表示します。（⇐⇒PRESET）

書　式

```
PSET(x,y [,パレットコード])
```

x, y……………… ドットの表示座標（論理座標）
　x……水平座標
　y……垂直座標
パレットコード…ドットのパレット（色を指定します）
　省略すると現在のパレットコード（COLOR文参照）が使われます。

省略形　PS.

解　説
グラフィック画面の論理座標（x,y）の点を指定のパレット番号でセットします。マルチ画面モードの時は、指定の画面のみパレットコード0ならリセット、それ以外ならセットします。
WINDOW外の点を指定してもエラーにはならず表示もされません。

サンプルプログラム
ドットの水平・垂直座標とカラーをランダムに指定して、画面に表示します。

```
LIST
10 ' PSET
20 INIT:WIDTH 40
30 X=INT(RND*320)
40 Y=INT(RND*200)
50 C=INT(RND*7)+1
60 PSET(X,Y,C)
70 GOTO 30
OK
```

止めるときは、SHIFT を押しながら BREAK を押すか
CTRL を押しながら C を押してください。

2.6.2 PRESET

機　　能　グラフィック画面の点（ドット）をリセットします。(⇐⇒PSET)

書　　式

```
PRESET(x,y[,パレットコード])
```

x,y……………… ドットの表示座標（論理座標）
x …水平座標
y …垂直座標
パレットコード…ドットの消去パレット

省 略 形　PRE.

解　　説
グラフィック画面の論理座標（x,y）の点を指定のパレットコードでリセットします。
グラフィック画面上の各ドットについて、表示されていれば（1ビットセットされていれば）消し、されていなければ何もしません。
マルチ画面モードでは0で何もされずにそれ以外では、リセットされます。

サンプル プログラム　白い画面の各ドットをランダムにリセットします。

```
LIST
10 INIT:WIDTH 40
20 LINE(0,0)-(319,199),PSET,7,BF
30 X=INT(RND*319)
40 Y=INT(RND*200)
50 C=CINT(RND*7+.5)
60 PRESET(X,Y,C)
70 GOTO 30
OK
```

2.6.3 LINE

機　能　画面上に直線や長方形を描きます。

書　式

```
LINE[(x1,y1)]-(x2,y2) { ,PSET   } { [,パレットコード]                    }
                      { ,PRESET } { ,[パレットコード],ラインパターン        }
                      { ,XOR    } { [,パレットコード],B[,ラインパターン]   }
```

省略形　なし。

解　説　論理座標 (x_1, y_1) - (x_2, y_2) 間に直線を描いたり2点を対角とする長方形を描いたりします。(x_1, y_1) を省略するとその前の命令の終点 (x_2, y_2) から新しい終点へと描きます。

モードは、セットする、リセットする、反転するの3つのモードがありそれぞれ、PSET，PRESET，XORと指定します。パレットコードは省略された場合、COLOR文の表示色と同じパレットコードが使われます。

モードの次か、パレットコードの次にBを指定すると長方形を描きます。Bがない場合は直線を描きます。

ラインパターンは16ビットの数値を指定しますが省略の場合は&HFFFF（実線）で描きます。

点線を描く時は&HCCCC、一点鎖線は&HFFCCを指定するとよいでしょう。

サンプルプログラム　画面上にランダムに直線を引きます。

```
LIST
10 ' LINE SAMPLE
20 INIT:WIDTH 40
30 X0=INT(RND*320)
40 Y0=INT(RND*200)
50 X1=INT(RND*320)
60 Y1=INT(RND*200)
70 C=INT(RND*7)+1
80 LINE(X0,Y0)-(X1,Y1),PSET,C
90 X0=X1:Y0=Y1
100 GOTO 50
OK
```

```
80 LINE(X0,Y0)-(X1,Y1),PSET,C,&HCCCC
```

点線を描く時の指定方法

```
80 LINE(X0,Y0)-(X1,Y1),PSET,C,&HFFCC
```

一点鎖線を描く時の指定方法

LINE

機能　画面上に長方形を描き，その内部をぬりつぶします。

書式

```
LINE[(x1,y1)]-(x2,y2) {,PSET   } {,BF
                      {,PRESET } {,[パレットコード],BF
                      {,XOR    } {,BF,タイルパターン
```

省略形　なし。

解説　論理座標 (x_1, y_1) - (x_2, y_2) の範囲内を指定のモードでぬりつぶします。ラインやボックスと同じパラメーターを持ちモードの次かパレットコードの次に、BFと指定する事によりボックスフルを実行します。

BFの次にタイルパターンを表わす文字式を指定する事によりそのパターンでのぬりつぶしが可能となります。(タイルパターンについては、PAINT参照)

サンプルプログラム

```
LIST
10 INIT:CLS0
20 LINE(0,0)-(100,100),PSET,3,BF
30 LINE(100,100)-(200,199),PSET,BF,HEXCHR$("AAAAAA555555")
OK
RUN
OK
```

LINE

機能　指定の論理座標間を実線でつなぎます。

書式

```
LINE[(x1,y1)][-(x2,y2)][-(x3,y3)…-(xn,yn)]
```

省略形　なし。

解説　指定の論理座標間を次のラインでつなぎます。

モード……………PSET。

パレット番号…COLOR文の表示色と同じ。

ラインパターン…&HFFFFの実線。

消したい時はCOLOR 0 を実行後この命令を実行します。

始点を省略した場合最後の終点から引きます。

また、始点のみを指定した場合最後の終点へ引きます。

サンプルプログラム

```
INIT:CLS0:LINE(100,0)-(0,100)-(100,200)-(200,100)-(100,0)
```

LINE

機　　能

テキスト画面に線や長方形を描きます。

書　　式

LINE[(x₁, y₁)]-(x₂, y₂), 文字式 { 省略 / ,B / ,BF }

x_1, x_2……テキスト上のX座標 (WIDTH40時のとき0 ～39 / WIDTH80時のとき0 ～79)

y_1, y_2……テキスト上のY座標（0 ～24）

省 略 形

なし。

解　　説

テキスト画面に対して、ラインボックス、ボックスフルの各命令を実行します。そのキャラクターは、文字式の先頭の文字が使われアトリビュートは、COLOR命令等で指定したものです。文字式がヌル（"")の時は、アトリビュートのみ描きます。

WINDOW命令で絶対座標と論理座標を別の座標系としている時は、その比率が影響しますので上記の座標系と異なります。

サンプル
プログラム

```
LIST
10 INIT:CLS4
20 LINE(1,24)-(38,24),"┴"
30 LINE(0,1)-(0,23),"├"
40 LINE(39,1)-(39,23),"┤"
50 LINE(1,0)-(38,0),"┬"
60 LINE(1,1)-(38,23),"┼",BF
90 GOTO 90
OK
```

2.6.4 CIRCLE

機　　能

円・弧を描きます。

書　　式

CIRCLE(x,y),r[,[パレットコード],[扁平率],[初期角]〔,終了角〕]

x,y………………… 中心の座標（論理座標）

　　x 水平座標

　　y 垂直座標

r …………………中心と円周との座標差（半径）

パレットコード…円弧の色（COLOR文参照）省略すると現在の色

扁平率……………（扁平率）＝（垂直方向の半径）÷（水平方向の半径）省略すると1

初期角……………0 ～360(度) 省略すると0(度)

終了角……………0 ～360(度) 省略すると360(度)

省 略 形

CI.

解　　説

画面にドットで円や弧を描くためのステートメントです。

中心（x,y)、半径rの円が画面に描かれます。扁平率を設定すると､x方向とy方向異なる半径の円を描くことができます。

初期角と終了角を指定すると、その内角の弧が表示されます。

WIDTH80の時のx方向の半径は2rとなります。

サンプル
プログラム

```
LIST
10 INIT:WIDTH 40
20 FOR X=20 TO 319 STEP 40
30 FOR Y=20 TO 199 STEP 40
40 C=INT(RND*7)+1
50 CIRCLE(X,Y),20,C,1,0,360
60 NEXT
70 NEXT
OK
```

2.6.5 POLY

機能	多角形を描きます。

書式

POLY(x,y),r[,[パレットコード],[ステップ角],[初期角][,終了角]]

x,y……………… 中心の座標（論理座標）
x 水平座標
y 垂直座標
r …………………中心から頂点までの論理座標差
パレットコード…多角形の辺の色（COLOR文参照）。省略すると現在の色
ステップ角……… 0 ～360(度)。 n角形のとき$\frac{360}{n}$(度)。省略すると1（度）
初期角…………… 0 ～360(度)。省略すると0（度）
終了角…………… 0 ～360(度)。 〃 360(度)

省略形 POL.

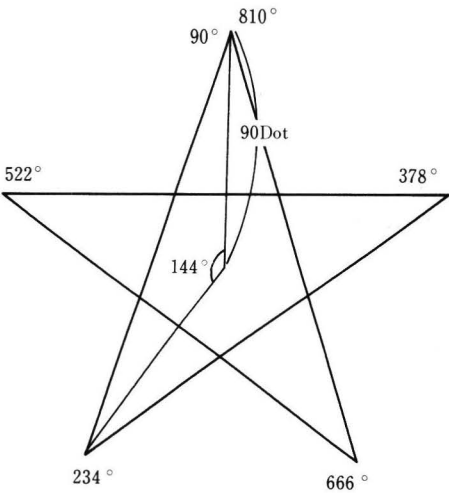
解説

(x,y) を多角形の中心とし、中心から各頂点までの距離をrとした多角形を描きます。頂点間の (x,y)を多角形の中心とし、中心から各頂点までの距離をrとした多角形を描きます。頂点間の角度をステップ角と呼び120で三角形、90で四角形となります。この角度が0の時は、中心から初期角の方向へ長さrの直線を引きます。

サンプルプログラム

星を描きます。

```
POLY(100,100),90,5,144,90,810
OK
```

半径90パレットコード5の色で144度きざみ90度から810度までです。

2.6.6 PAINT

機　　能　画面の特定のエリアを指定した色で塗りつぶします。

書　　式

```
PAINT(x,y) ⎧,パレットコード          ⎫
           ⎨,タイルパターン [,境界のパレットコード,…]⎬
           ⎩  省略                      ⎭
```

省 略 形　PAI.

解　　説

論理座標（x,y）の点を塗り始めの点として、境界となるパレットコードで指定された図形内を、指定のパレットコードあるいはタイルパターンで塗りつぶします。

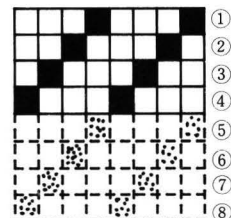
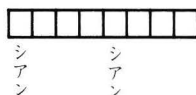
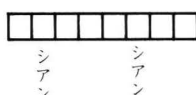
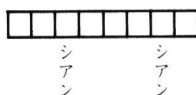
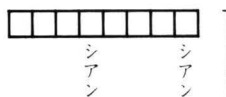
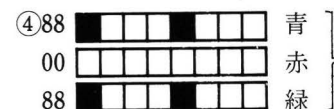
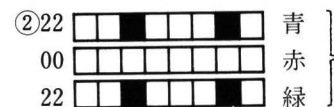
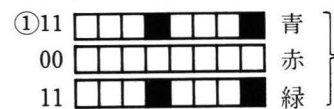
省略の場合、前のグラフィック命令のパレットコードが使われます。

タイルパターンは、横方向に8ドット、縦方向に任意の長さをもつタイル模様です。縦方向nドットのタイルを指定するためにはn×3文字の文字列が必要で、3文字で横8ドット縦1ドットの最小パターンを指定し、左から1文字ずつ青、赤、緑のドットパターンを示します。下に8×1ドットのタイルパターンを示します。

（例）

次の例は8×8ドットのタイルパターンです。

（例）

11＝（00010001）$_2$
22＝（00100010）$_2$
44＝（01000100）$_2$
88＝（10001000）$_2$
00＝（00000000）$_2$

サンプル
プログラム

```
CLS0
OK
PAINT(0,0),2
OK
```

画面を消し左上スミ（0,0）から2の色でぬりつぶします。
境界が無いので全画面ぬりつぶします。

```
CIRCLE(123,85),67,4
OK
PAINT(123,85),2,4
OK
```

境界線となる閉鎖空間（たとえば円など）を作ってそのぬりつぶす始めの点は、円の中心にするとまちがいありません。
ぬりつぶす色の次に境界線の色です。円は4の色で描いてあるので4に指定するとよいでしょう。
タイルパターンに重ねてタイルパターンでぬりつぶす場合、途中で止まってしまったり、Out of memoryエラーが出る場合がありますので、事前に単色でぬりつぶしてから、タイルパターンでPAINTする様にして下さい。

```
PAINT(123,85),HEXCHR$("11001122002244004488008 8"),4
OK
```

2.6.7 POSITION

機能

PATTERN文によるドットパターンの表示位置を指定します。

書式

POSITION x,y

x,y……ドットパターンの表示位置の座標
x…ドットパターンの表示位置の水平座標
WIDTH40のとき　0～319
WIDTH80のとき　0～639
y…ドットパターンの表示位置の垂直座標
0～199

省略形

POS.

解説

このステートメントはPATTERN文によって設定されるドットパターンの開始座標パターンの左上隅の座標）を指定し、必ずPATTERN文の前に置かれます。

サンプル
プログラム

PATTERN文参照。

```
POSITION 10,20
OK
```

2.6.8 PATTERN

機　能　グラフィック画面の特定のエリアに任意のグラフィックパターンを描きます。

書　式

```
PATTERN n, 文字列[, 文字列…]
```

n…………ドットパターンの垂直方向の段数 -255～255
　　n＜0のとき　下方向に重なる段数
　　n＞0のとき　上方向に重なる段数
　　n＝0のとき　Illegal function call を起こします。
文字列……ドットパターン（ビットパターン）を表す文字列。

省略形　PAT.

解　説

POSITIONで指定された座標を開始点として、ストリングで表わされたビットパターンを、画面にn段分（垂直方向にnドット分）表示します。

ストリングをいくつかに分けて指定すると、画面にはドットパターンが連続して表示されます。

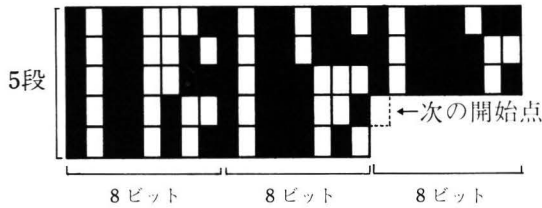
POSITIONが1度しか定義されていないで、その後PATTERN文が続く場合は表示位置を右か下へずらして、グラフィックパターンを表示し続けます。

描くべきグラフィックパターンは1バイト（8ビット）単位のドットパターンから構成されていて、ドットパターンは各セット・リセットの組み合わせにより256通りあります。ドットパターンを表すデータには文字列を使います。

たとえば■□■□■□■□(■はセット、□はリセット)を表示するには、$(10101010)_2=(AA)_{16}=(170)_{10}$ を文字に変換したCHR$(&HAA)かCHR$(170)、またはカタカナの小文字のエ "ェ" を使い、□■□■□■□■を表示するには、$(01010101)_2=(55)_{16}=(85)_{10}$ を文字に変換したCHR$(&H55)かCHR$(85)、あるいは "U" を使います。

ドットパターンの積み重ねはnの値の符号によって決められ、nが負のとき、上から下へ、nが正のとき下から上へ積み重ねられます。文字列データが積み重ねの段数より多いときは、下方向、上方向にn段積み重ねたあと、すぐ右隣り（x軸正方向）で同じ方向に積み重ね、すべての文字列データを表示しつくすまで同じ操作を繰り返します。

たとえば、A＄＝"アイウエオカキクケコサシス"
すなわち、HEXCHR$（"B1B2B3B4B5B6B7B8B9BABBBC"）
または、　CHR$（&HB1）＋CHR$（&HB2）＋……＋CHR$（&HBC）
をPATTERN文を使って画面に表示すると、下の図のようなドットパターンとなります。

PATTERN -5, A$

サンプルプログラム

上の例のサンプルプログラムを示します。

```
LIST
10 INIT:CLS4
20 A$="アイウエオカキクケコサシス"
30 POSITION 0,20
40 PATTERN -5,A$
OK
```

下にPATTERN文によって漢字を表示させる1例を上げます。実行させると、画面左上に「漢」という字が表示され、続いて右隣りに同じ「漢」が表示されます。

```
LIST
10 INIT:CLS4
20 POSITION 0,20
30 PATTERN -16,KANJI$(&H7F1)
40 PATTERN -16,KANJI$(&H7F1)
OK
```

KANJI$についてはKANJI$関数を参照してください。

注）専用漢字ROMのついていない場合は16×16ドットの白い四角形が2つ続いて表示されます。

2.7 画面制御ステートメント グラフィックステートメント}の使用可能な画面

●CONSOLE
(文字の表示エリアの設定)
●LOCATE,CURSOR
(カーソルの位置の指定)
●CREV
(文字の反転表示)
●CFLASH
(文字のフラッシング表示)
●CGEN
(ユーザ文字モードの設定)
●CSIZE
(文字のサイズの指定)
●DEFCHR$
(キャラジェネに文字パターンを定義)

●PALET
(パレットカラーの指定)
●OPTION SCREEN
(グラフィック用メモリーの用途設定)

〈テキスト画面〉
Text 8色

〈グラフィック画面〉
G1 青
G2 赤
G3 緑

●WIDTH
(画面サイズの設定)
●SCREEN,GRAPH
(I/Oページ、グラフィックモードの設定)
●COLOR
(カラー設定)
●CLS
(画面のクリア)
●GET@
(ドットデータの読み込み)
●PUT@
(ドットデータの表示)
●POKE@
(VRAM内へドットデータを書き込む)
●LINE
(直線、長方形の表示)

●WINDOW
(グラフィック表示エリアの設定)
●PSET
(ドットの表示)
●PRESET
(ドットのリセット)
●CIRCLE
(円、弧の表示)
●POLY
(多角形の表示)
●PAINT
(指定エリアの塗りつぶし)
●POSITION
(グラフィックパターンの表示位置の指定)
●PATTERN
(グラフィックパターンの表示)

●PRW
(Text/グラフィックの優先順位の設定)
●LAYER
(Text/G1/G2/G3の優先順位の設定)

〈マルチ画面〉
G1 8色
G2 8色
G3 8色

●CANVAS
(グラフィックのカラー指定)

2.8 特殊ステートメント

2.8.1 KEY・DEF KEY

機　能　ファンクションキーに対して文字列を定義します。

書　式

KEY ファンクションキー番号，文字列

DEF　KEY ファンクションキー番号，文字列

ファンクションキー番号……1～10（ファンクションキー左端から1,2,3,4,5,SHIFTキーを押しながら押すと6,7,8,9,10と対応しています。）

文字列……………………………最大15文字まで定義することができ、15文字を越えるとはみ出した右側の文字が無視されます。

省略形　K.

DEFK.

解　説　ファンクション番号で指定したキーに文字列を対応させます。このステートメントによって文字列を定義すると、それ以降、そのキーを押すと定義した文字列をキーボードから打ち込むのと同じになります。

ファンクションキーはBASICロード時には次のように定義されています。

1．"AUTO"+CHR$(13)
2．"?TIME$"+CHR$(13)
3．"KEY"
4．"LIST"+CHR$(26,13)
5．"RUN　"+CHR$(13)
6．"LOAD "+CHR$(13)
7．"WIDTH "
8．"CHR$("
9．"PALET "
10．"CONT"+CHR$(13)

2.8.2 KEYLIST・KLIST

機　能　ファンクションキーの定義状態を表示します。

書　式

{KEY　LIST / KLIST}

省略形　K.L.

KL.

解　説　このステートメントを実行すると、画面にキー番号と文字列の一覧表を表示します。

サンプルプログラム

```
KEYLIST

KEY 1,"AUTO"+CHR$(13)
KEY 2,"?TIME$"+CHR$(13)
KEY 3,"KEY"
KEY 4,"LIST"+CHR$(26,13)
KEY 5,"RUN   "+CHR$(13)
KEY 6,"LOAD  "+CHR$(13)
KEY 7,"WIDTH  "
KEY 8,"CHR$("
KEY 9,"PALET  "
KEY10,"CONT"+CHR$(13)
OK
```

2.8.3 KEY0

機能 キーボードから入力のシミュレートを行う。

書式

KEY0 ,文字列

省略形 K. 0

解説 プログラムの入力命令の前に置いて使います。入力命令に来ると、ユーザーがキーボードのキーを押さなくても、指定した文字列が自動的に入力されます。
なお、64文字目から無視します。

サンプルプログラム

```
KEY0,"FORI=0TO10:?I;:N."+CHR$(13)   この行を入力すると、
OK
FORI=0TO10:?I;:N.                   自動的にこの行が入力されて、
 0  1  2  3  4  5  6  7  8  9  10   実行されます。
OK
```

2.8.4 REPEAT ON・REPEAT OFF

機能 キー入力のリピート機能の設定、解除を行います。

書式

REPEAT {ON / OFF}

省略形 REP.O.
REP. OF.

解説 通常、キーボードのキーを押し続けると、同じ文字が継続して入力されますが、この機能を止めるのがREPEAT OFFです。
REPEAT OFFにすると、キーボードからのキー入力が1文字で止まり、それ以上受けつけなくなります。REPEAT ONすると元の状態に戻ります。

サンプルプログラム

```
REPEAT ON
OK
REPEAT OFF
OK
```

2.8.5 CLICK ON・CLICK OFF

機能 キー入力時のクリック音の設定、解除を行います。

書式

```
CLICK {ON
      {OFF
```

省略形 CLI.O.

CLI.OF.

解説 キーボードのキーを押すと「クリッ」という音（クリック音）を出しますが、CLICK OFFにするとこの音を出しません。CLICK ONにすると、再びクリック音が出るようになります。

サンプルプログラム

```
CLICK ON
OK
CLICK OFF
OK
```

2.8.6 ON KEY GOSUB

機能 ファンクションキーからの入力値によっていくつかの行へジャンプします。

書式

```
ON KEY GOSUB　行番号1〔，行番号2 ，行番号3 ，…〕
```

省略形 O.　K.　GOS.

解説 i番目のファンクションキーが押されたとき、i番目に指定した行番号の行をサブルーチンとして実行します。

ファンクションキー1ならば1番目の行番号，2ならば2番目，3ならば3番目，…というように対応しており、ファンクションキーの値に対応する行番号がなければ次の文へ実行を移します。

ファンクションキーは左から1，2，……，5、シフトを押しながらのときは6，7，……，10の値をとります。

ON KEY GOSUBを実行すると本来のファンクションキーの機能である文字列定義の機能が使用できなくなります。(CTRL ＋ D を押すとON KEY GOSUBは解除されます。)

サンプルプログラム

```
LIST
10 ON KEY GOSUB 100,200
20 PRINT"*";:GOTO 20
100 BEEP:RETURN
200 CLS:RETURN
OK
```

このプログラムを実行しますと画面上に星印を書きますが、ファンクションキー1を押すとBeep音が鳴り、ファンクションキー2を押すと画面がクリアされて実行が継続されます。

SHIFT ＋ BREAK でプログラムは止まりますが、その後、ファンクションキーを押しますと出力されませんので CTRL ＋ D を押してください。

2.8.7 KEY ON・KEY OFF・KEY STOP

機　　能　ON KEY GOSUB命令を制御します。

書　　式

```
KEY〔n〕 ┌ON  ┐
         │OFF │
         └STOP┘
```

n…1～10の整数。

省 略 形　K.O.
K.OF.
K.S.

解　　説　n番目のファンクションキーについて
KEY n ONの場合、押されたときON KEY GOSUBを実行します。
KEY n OFFの場合、押されたとき本来のファンクションキーが押されたものとみなします。
KEY n STOPの場合、押されたとき、ON KEY GOSUBは実行せず、KEY n ONが実行されるまで、保留しています。
nを省略すると、1～10すべてのファンクションについて上記の機能をもちます。

サンプル プログラム

```
KEY ON
OK
KEY OFF
OK
KEY STOP
OK
```

2.8.8 MID$

機　　能　文字列の一部の置き換えをします。

書　　式

```
MID$(文字列1 ，m ，n)＝文字列2
```

文字列1……置き換えされる文字列、結果になる。
m……………文字列1の置き換え開始位置。
n……………文字列1の置き換える文字列の長さ。
文字列2……置き換わる文字列。

省 略 形　MI.

解　　説　文字列1のm番目の文字からn個の文字を、文字列2の最初からn個の文字で置き換えます。
mの値が文字列1の長さを越えた場合は何もしません。nの値が文字列2の長さより大きくなると、nは文字列2の長さになります。

サンプル プログラム

```
LIST
10 A$="ABCDEFGHIJKLMN"
20 MID$(A$,2,3)="1234"
30 PRINTA$
OK
RUN
A123EFGHIJKLMN
OK
```

2.8.9 MEM$

機　能　指定されたメモリーへ、文字列を転送します。

書　式

```
MEM$(アドレス，n)＝文字列
```

アドレス……メインメモリー内の先頭アドレス(0～&HFFFF)。
n……………転送される文字数(0～255)。
文字列………転送しようとする文字列。

省略形　MEM.

解　説　指定したアドレスから始まるメインメモリー内の領域にn個の文字を転送します。

サンプルプログラム

```
MEM$(&HFF00,5)="12345"
OK
?MEM$(&HFF00,5)
12345
OK
```

2.8.10 MON

機　能　制御をBASIC内蔵の機械語モニターに移します。

書　式

```
MON
```

省略形　MO.

解　説　この命令は、BASIC内蔵の機械語モニターを起動します。
この機械語モニターについては「モニターのコマンド」を参照してください。

サンプルプログラム

```
MON
*D 0000 003F
:0000=C3 FA 00 C3 7C 01 50 28 /テ.テ|.P(
:0008=C3 D4 03 C3 84 04 18 05 /テヤ.テ■.%.
:0010=C3 D4 03 C3 BD 04 00 18 /テヤ.テス...
:0018=C3 D4 03 C3 9D 02 00 27 /テヤ.テ...'
:0020=C3 D4 03 C3 07 0E 07 20 /テヤ.テ...
:0028=C3 D4 03 C3 AA 0A 0D B7 /テヤ.テエ..キ
:0030=C3 D4 03 C3 CA 0A 00 FF /テヤ.テハ..テ
:0038=C3 D4 03 C3 75 0B C3 79 /テヤ.テu.テy
*R
OK
```

2.8.11 HCOPY

機　能　画面をラインプリンターに出力します。

書　式

```
HCOPY〔n〕
```

n…0～4の整数

省略形　H.

解　説　この命令は画面をラインプリンターにコピーするものであり、nの値は0～4です。

nの値

0　グラフィック1，2，3全画面を合成したものをコピーします。
1　グラフィック1の画面をコピーします。
2　グラフィック2の画面をコピーします。
3　グラフィック3の画面をコピーします。
4　テキスト画面と、グラフィック1，2，3全画面を合成したものをコピーします。
省略　テキスト画面をコピーします。

（注意）
テキスト画面の中で、CSIZE0以外の倍文字キャラクターはコピーできません。また、ユーザーキャラクタジェネレーターは青・赤・緑の各パターンの合成されたものとなります。

サンプルプログラム

```
HCOPY
OK
HCOPY 0
OK
```

コピーされた文字のサイズは下のようになります。

普通の活字のサイズ	**コピー文字のサイズ**	
	WIDTH40のとき	WIDTH80のとき
ABC	ABC	ABC

2.8.12 BOOT

機　能　IPL,を起動します。

書　式

```
BOOT
```

省略形　BO.

解　説　このコマンドを実行したら、その前の状態には戻れません。
あくまでもIPLの管理下になります。

サンプルプログラム

```
BOOT
Are you sure ? (y or n)N
Ok
```

BOOTとキーボードから打ち込むと「Are you sure ? (y or n)」と表示され、キー入力を持っているので、実行したければYキーを押し、中止したければそれ以外のキーを押してください。Yキーを押すと、電源投入直後の状態になり、プログラムがすべて消去されます。

2.8.13 PAUSE

機　能　プログラムの実行を一時休止します。

書　式

```
PAUSE n
```

n……0～255

省略形　PA.

解　説　このステートメントを実行すると、
設定時間分プログラムの実行を休止して、次に進みます。休む時間は1単位につき約0.1秒です。

サンプルプログラム

```
TIME=0:PAUSE 100:?TIME
 10
Ok
```

10秒の空回り

2.8.14 WAIT

機　能　ポートからの条件待機をします。

書　式

```
WAIT ポート番号、式1〔,式2〕
```

ポート番号……&H0000～&HFFFF
式1，2……整数

省略形

解　説　(ポート番号で指定した)入力ポートから読み込んだデータと式2とのエクスクルーシブオァ(exclusive or「論理演算子」)をとり、その結果と式1とのANDをとります。そしてもし結果が0のときは、もう1度入力ポートからのデータを読み込んで、前述と同じ操作を繰り返します。結果が0でないときはWAITの次の文へ実行を移します。
このようにして、WAIT文は入力ポートが、ある状態になるまで、プログラムの実行を停止して待ちます。
また、定数2を省略すると定数2には0がセットされます。

サンプルプログラム

```
WAIT &HC0,I,J
Break
Ok
```

2.9 タイマー制御ステートメント

2.9.1 TIME$

機　能　時間の設定をします。

書　式

TIME$＝"hh：mm：ss"

h h……時間 00～23
m m……分　00～59
s s……秒　00～59

省略形　なし。

解　説　ユーザーが時間を設定するのに使うステートメントです。

サンプルプログラム

```
TIME$="12:34:56"
OK
?TIME$
12:34:57
OK
```

2.9.2 DAY$

機　能　曜日の設定をします。

書　式

DAY$＝"XXX"

省略形　XXX……曜日

DAY.

解　説　ユーザーが曜日を設定するときに使うステートメントです。

SUN　日曜日
MON　月 〃
TUE　火 〃
WED　水 〃
THU　木 〃
FRI　金 〃
SAT　土 〃

サンプルプログラム

```
DAY$="SUN"
OK
?DAY$
SUN
OK
```

2.9.3 DATE$

機　能　年月日の設定をします。

書　式

```
DATE$="yy/mm/dd"
```

yy……年　00～99
mm……月　01～12
dd……日　01～31

省略形　DATE.

解　説　ユーザーが年月日を設定するときに使うステートメントです。初期状態で年は82がセットされますので設定をしなおす必要があります。

サンプルプログラム

```
DATE$="82/12/29"
OK
?DATE$
82/12/29
OK
```

2.9.4 TIME

機　能　秒数の設定をします。

書　式

```
TIME=X
```

X…秒数　0～86399

省略形　TIM.

解　説　ユーザーが秒数を設定するのに使います。

サンプルプログラム

```
LIST
10 PRINT TIME$
20 TIME=0
30 FOR I%=0 TO 10000:NEXT
40 PRINT TIME
50 PRINT TIME$
OK
RUN
14:16:08
 3
14:16:11
OK
```

2.9.5 ASK

機能 会話型のタイマー設定ルーチンを呼び出します。

書式

```
ASK
```

省略形 AS.

解説 会話タイプのタイマーの設定ルーチンを呼び出します。これによって、7つのタイマーのセットができます。ただし、WIDTH40のSCREEN 0 , 0 に設定されている必要があり、それ以外のときASKを実行するとBad screen modeエラーが出ます。

サンプルプログラム

ASK ⏎ と入力すると、画面に下の図が表示されますので画面メーセージに従ってタイマーを設定してください。

年は設定できますが変化しません。初期状態では82年が設定されます。（ウルウ年の判別はできませんので、ウルウ年がある場合は再設定必要です。）

```
TV Timer control

     82/11/10 WED 10:34:54

TIMER1   XX/XX XXX XX:XX    OFF
TIMER2   XX/XX XXX XX:XX    OFF
TIMER3   XX/XX XXX XX:XX    OFF
TIMER4   XX/XX XXX XX:XX    OFF
TIMER5   XX/XX XXX XX:XX    OFF
TIMER6   XX/XX XXX XX:XX    OFF
TIMER7   XX/XX XXX XX:XX    OFF

          Month  01 - 12 or XX

[ESC]=Exit, [CLR]=Reset, [CR]=Set
```

2.10　テレビ制御ステートメント

専用ディスプレイテレビCZ-800Dにのみ有効な機能です。

2.10.1　TVPW

機　能　テレビの電源のon/offを行います。

書　式

```
TVPW { ON  }
     { OFF }
```

省略形　TV.O.
TV.OF.

解　説　専用テレビの電源のスイッチを入れたり切ったりする命令です。ONのときはテレビのモードなので、コンピュータのディスプレイとして使用するときはCRT(CRT文参照) でモードを設定する必要があります。

サンプルプログラム

```
TVPW ON:CRT 1
Ok
```

(注意)　連続してテレビの電源スイッチを入れたり切ったりすることは、テレビ故障の原因となりますので、絶対にやめてください。

2.10.2 CRT

機　　能　画面表示の設定をします。

書　　式

```
CRT n
```

n＝0，1，2，3

省 略 形　なし。

解　　説　このステートメントの実行によって画面表示の切り換えを行うことができます。

nの値

0　テレビ放送を表示します。

1　コンピュータ画面を表示します。

2　テレビ放送のコントラストを下げて、コンピュータ画面を重ねて表示します。

3　テレビ放送とコンピュータ画面を同時に重ねて表示します。

サンプルプログラム

```
CRT 2
OK
```

2.10.3 CHANNEL

機　　能　テレビのチャンネルを変えます。

書　　式

```
CHANNEL n
```

n…テレビのチャンネル。1～12。

省 略 形　CHAN.

解　　説　テレビ放送のチャンネルを指定します。

サンプルプログラム

```
CHANNEL 12
OK
CHANNEL 4
OK
```

2.10.4 VOL

機　能	テレビの音量を変えます。
書　式	VOL[n]

n＝-1，0，1

省略形	VO.

解　説

引数を省略すると、標準の音量、
n＝-1のとき、音量はダウン、
n＝1　のとき、音量はアップ、
n＝0　のとき、ミュートのon/offが反転します。

サンプルプログラム

```
VOL 1
OK
VOL-1
OK
VOL 1
OK
VOL 0
OK
```

2.10.5 SCROLL

機　能	画面のスクロールを行う。
書　式	SCROLL n

n＝-3～3までの整数

省略形	SCRO.

解　説

テレビ放送とコンピュータ画面が重なった状態のとき（CRT2、CRT3のとき)、コンピュータ画面の上下方向のスクロールを行います。

nの値が正の数のときは上方向に、負の数のときは下方向にスクロールし、その速さは上下ともに3段階に設定されていて、nの絶対値が大きいほど速くスクロールします。nの値が0のとき、または省略すると、スクロールはストップします。

サンプルプログラム

```
SCROLL -1
OK
SCROLL 1
OK
SCROLL 0
OK
```

2.11 カセット制御ステートメント

2.11.1 EJECT

機　能　カセットのドアを開けます。

書　式

```
EJECT
```

省略形　EJ.

解　説　カセットのドアを開けて、画面は入力待ちの状態になります。

カセットがFAST(FAST文参照),REW(REW文参照)の状態にあるときは、CSTOP(CSTOP 文参照)が行われてから、EJECTが実行されます。

サンプルプログラム

```
EJECT
```

2.11.2 CSTOP

機　能　カセットテープの走行をストップします。

書　式

```
CSTOP
```

省略形　CST.

解　説　テープ走行をストップさせます。FAST(FAST文参照),REW(REW文参照)などの命令により、カセットが作動しているときそれを止める命令です。ただし、LOAD(LOAD文参照)やSAVE(SAVE文参照)をしているとき、あるいはAPSS(APSS文参照)の途中では実行しません。

サンプルプログラム

```
CSTOP
```

2.11.3 FAST

機　能　カセットテープ走行の早送りをします。

書　式

```
FAST
```

省略形　FA.

解　説　カセットテープ走行を早送りの状態にします。カセットテープが入っていない場合は何もしません。

サンプルプログラム

```
FAST
```

2.11.4 REW

機能 カセットテープの巻き戻しをします。

書式

```
REW
```

省略形 なし。

解説 カセットテープを巻き戻しの状態にします。カセットテープが入っていない場合は何もしません。

サンプルプログラム

```
REW
```

2.11.5 APSS

機能 カセットテープのプログラムの頭出しをします。

書式

```
APSS n
```

n …… −50〜50の整数

省略形 AP.

解説 カセットテープのプログラムの頭出しをする命令で、nが0のときは何もせず、正の数のときFAST方向の頭出しを、負の数のときREW方向の頭出しをします。

ダイレクト実行のAPSS中は、残り1ブロックになると、画面上でカーソルが点滅を始めますが、キー入力は受けつけられません。ただし、押されたキーは7文字まで記憶されるので、APSSが終わりしだいエコーバックされます。SHIFT ＋ BREAK キーか、カセットキーを押すことにより途中で実行を止めることができます。

サンプルプログラム

```
APSS -1
```

2.11.6 CMT

機　能　カセットの状態の設定を行います。

書　式

```
CMT＝n
```

n……0～6,10

省略形　CM.

解　説　nの値によってカセットの状態を設定します。

nの値	カセットの状態
0	EJECT
1	CSTOP
2	PLAY
3	FAST
4	REW
5	APSS＋1
6	APSS－1
10	RECORD

2～10はカセットが入っていないと何もしません。
また、10はプログラムのツメが折ってあると何もしません。
カセットテープの消去はCMT＝10として下さい。

サンプルプログラム

```
FAST
OK
?CMT
 3
OK
REW
OK
?CMT
 4
OK
CSTOP
OK
?CMT
 1
OK
```

2.12 サウンド制御ステートメント

2.12.1 BEEP

機　能　「ポッ」という音を出します。

書　式

```
BEEP { 0 | 1 | 省略 }
```

省略形　BE.

解　説　このステートメントは「ポッ」という音を1回だけ出すためのものです。

パラメータが1のとき、チャンネルCから音を出します。

パラメータが0のとき、チャンネルA、B、Cすべての音を止めます。

パラメータを省略すると、BEEP1とBEEP0が続いて実行されたのと同様の処理が行われ、1回だけ音を出して止まります。

サンプルプログラム

```
BEEP 1
OK
BEEP 0
OK
BEEP
OK
```

2.12.2 MUSIC

機　能　音楽の演奏をします。

書　式

```
MUSIC ストリングデータ
```

ストリングデータ…以下で説明する文字列変数をストリングデータといいます。

省略形　MU.

解　説　ストリング式で指定された楽譜に従って演奏します。ストリング式は、音程オクターブ、音の長さ、音量、和音を指定するもので、以下に説明する文字を並べたものです。

①音程…音程は下記のようにC～Bの文字を使って表されます。

音　程	指定方法
ド	C
ド#(レb)	#C
レ	D
レ#(ミb)	#D
ミ	E
ファ	F
ファ#(ソb)	#F
ソ	G
ソ#(ラb)	#G
ラ	A
ラ#(シb)	#A
シ	B

②和音…２重音や３和音を演奏するには、上声部と下声部の間にチャンネルセパレータ：をはさみます。
"チャンネルＡ：チャンネルＢ：チャンネルＣ"
③長さ…音程の後に０～９の整数をつけて、音の長さを指定します。

音　の　長　さ	対応する整数
$\frac{1}{8}$（32分音符　）	0
$\frac{1}{4}$（16分音符　）	1
$\frac{3}{8}$（付点16分音符）	2
$\frac{1}{2}$（８分音符　）	3
$\frac{3}{4}$（付点８分音符）	4
1（４分音符　）	5
$1\frac{1}{2}$（付点４分音符）	6
2（２分音符　）	7
3（付点２分音符）	8
4（全音符　）	9

注）音の長さは４分音符（整数５）を１としたときの相対的な値です。
整数の指定がない場合は前の音と同じ長さを意味します。
④休符…Ｒ０～Ｒ９で、休みの長さを指定します。
⑤オクターブ…Ｏ１～Ｏ８でオクターブの指定をします。
また、音符の前に＋をつけて１オクターブ上の音程を、－をつけて１オクターブ下の音程を表します。
⑥音量…Ｖ０～Ｖ15で音量の指定をします。

サンプル
プログラム

オクターブ4のドからオクターブ5のドまでドレミファソラシドを出力します。

```
MUSIC"V1204CDEFGAB05C"
OK
```

オクターブ4ボリューム13でドミソの和音（コードC）を順に重ねていきます。

```
MUSIC"V1304CDEFG:V1204EFGAB:V1204GAB05CD"
OK
```

2.12.3 TEMPO

機　　能　音楽演奏のテンポを指定します。

書　　式

```
TEMPO  n
```

n……30～7500まで整数

省 略 形　TE.

解　　説　MUSIC文によって演奏される曲のテンポを指定します。指定された整数は1分間あたりの4分音符の拍数を示します。

サンプルプログラム

```
TEMPO 120
```

```
TEMPO 1600
```

2.12.4 PLAY

機　　能　音楽を演奏します。

書　　式

```
PLAY { n
     { ストリングデータ }
```

n……30～7500

ストリングデータ…（MUSIC文参照）

省 略 形　PL.

解　　説　MUSIC文とTEMPO文の両方の機能を持ち、PLAYの後ろが整数か数式の場合TEMPO文と同様の、ストリングデータ（文字式）の場合MUSIC文と同様の機能を持ちます。

サンプルプログラム

```
PLAY"V1204CDEFGABO5C"
PLAY"V1304CDEFG:V1204EFGAB:V1204GABO5CD"
```

2.12.5 SOUND

機　能　効果音を出します。

書　式

```
SOUND レジスター番号, 8ビットデータ
```

　レジスター番号……0 ～13
　8 ビットデータ……0 ～255

省略形　SO.

解　説
　サウンドジェネレータ用LSI(AY-3-8910) へ直接データを送り込むことによって音を出します。下の図でレジスターとデータとの関係を示します。

レジスター番号	ビット / レジスターの内容	7	6	5	4	3	2	1	0
0	チャンネルAの周波数（音程）	8ビット FTA							
1		未使用				4ビット CTA			
2	チャンネルBの周波数（音程）	8ビット FTB							
3		未使用				4ビット CTB			
4	チャンネルCの周波数（音程）	8ビット FTC							
5		未使用				4ビット CTC			
6	ノイズの周波数	未使用			5ビット データ				
7	チャンネルAの音量	IN/OUT		ノイズ			トーン		
		IOB	IOA	C	B	A	C	B	A
8	チャンネルAの音量	未使用			M	L3	L2	L1	L0
9	チャンネルBの音量	未使用			M	L3	L2	L1	L0
10	チャンネルCの音量	未使用			M	L3	L2	L1	L0
11	エンベロープ周期	8ビット FT							
12		8ビット CT							
13	エンベロープ形状	未使用				E3	E2	E1	E0

サンプルプログラム

```
SOUND 0,16:SOUND 1,2:SOUND 7,62:SOUND 8,15
OK
```

上のプログラムをダイレクトで実行すると、ブザーが鳴り始めます。
ブザーを止めるときは CTRL と D キーを同時に押してください。

2.13 書式指定（PRINT USING）

PRINT USINGによる書式指定は、次の書式指定子によってコントロールします。書式指定子には、数値型と文字型の2種類あります。

2.13.1 数値型

（1）#

表示したい最大の桁数を指定し、指定した桁数より数値の桁数が小さいときには右つめで表示されます。

```
LIST
10 A=2345                              (6桁の指定)
20 PRINT USING"######";A
OK
RUN
  2345
OK
```

（2）．

固定小数点の表示において、小数点の位置をそろえる指定で、小数点以下の桁指定も行えます。

```
10 A=23.456
20 PRINT USING"###.##";A               (小数点以下第2位)
OK
RUN
 23.46
OK
```

（3），

数値を3桁ごとに区切って、そこに,（カンマ）を入れて右づめで表示します。

```
10 A=23456789!
20 PRINT USING"###,###,###";A          (3桁ごとにカンマをつける)
OK
RUN
 23,456,789
OK
```

（4）＋と－

＋と－は、数値を表示するときに＋と－の符号をどこに表示するか指定します。

＋は#の前と後に、－は#の後のみに指定できます。

＋を指定した場合、正数の＋記号を出力します。

```
10 A=235:B=-440
20 PRINT USING"####+";A;B
30 PRINT USING"####-";A;B
40 PRINT USING"+####";A;B
OK
RUN
 235+ 440-
 235  440-
 +235 -440
OK
```

(5) ＊＊

桁指定をし、その範囲内でスペースがある場合、すべて＊で埋めるという指定をします。

＊自身を桁数に含まれます。

```
10 A=235
20 PRINT USING"**#####";A
OK
RUN
***235
OK
```

(6) ￥￥

数字の前に￥（円マーク）をつけます。￥自身も桁数に含まれます。

```
10 A=235
20 PRINT USING"¥¥#####";A
OK
RUN
   ¥235
OK
```

(7) ＊＊￥

この指定は、＊＊と￥￥を組み合せたもので、数字の前に￥をつけ、その範囲内でスペースがある場合、＊（アスタリスク）で埋めます。

```
10 A=235
20 PRINT USING"**¥####";A
OK
RUN
***¥235
OK
```

(8) ＾＾＾＾

桁数指定の＃の後ろに置くと、E表現、D表現のいずれも表示できるようになります。

```
10 A=235:B#=.00345#
20 PRINT USING"##.####^^^^";A;B#
OK
RUN
 2.3500E+02 3.4500D-03
OK
```

(9) 1つのUSINGで複数の変数を記述すると、その変数すべてに対し、USING指定したとみなします。

```
10 PRINT USING"####.###";101;2;10;-2;5.5;7.4
OK
RUN
 101.000    2.000   10.000   -2.000    5.500    7.400
OK
```

2.13.2　文字型

（1）！

！（感嘆符）を指定すると、文字型変数にある文字列の最初の1文字のみ表示します。

```
LIST
10 A$="123":B$="456"
20 PRINT USING"!";A$;B$
OK
RUN
14
OK
```

（2）&　　　&

指定文字列を&と&のスペースの個数＋2個分表示します。文字は、左づめで、スペースの個数＋2個以下のときは残りの右の部分をスペースで埋めます。

```
10 A$="123":B$="ABCDEF"
20 PRINT USING"&  &";A$;B$
OK
RUN
123 ABCD
OK
```

2.13.3　その他の型

＿（アンダーバー）

この後ろに伴った書式指定子1個を、単なる1文字とみなして表示します。

```
10 PRINT USING"_####";10
OK
RUN
# 10
OK
```

2.13.4　文字定数の出力

USINGの書式指定子のほかの文字がある場合は、それをそのまま表示します。

```
10 A=235:B=9.5
20 PRINT USING"DATA  ###";A
30 PRINT USING"##.## Sec";B
OK
RUN
DATA  235
 9.50 Sec
OK
```

第3章　関数・システム変数

3.1 数値関数

3.1.1 SIN

機　　能　数式のサインを与えます。

書　　式

```
SIN (x)
```

x…サインを求める数式。単位はラジアンです。

省 略 形　SI.

解　　説　数式のサイン、sin Xが、この関数の値になります。

πの値を3.14、PAI(1)、RAD(180)としたときのそれぞれについて、sin πの値を求めてみましょう。

サンプルプログラム

```
LIST
10 PA=3.14
20 PB=PAI(1)
30 PC=RAD(180)
40 PRINT SIN(PA)
50 PRINT SIN(PB)
60 PRINT SIN(PC)
OK
RUN
 1.5926531E-03
 0
 9.3132257E-10
OK
```

1.5926531E－03は1.5926531×10^{-3}

9.3132257E－10は9.3132257×10^{-10}

という意味で、どちらもほとんど0に近い値といえます。

PAI関数、RAD関数については、その関数の項を参照してください。

3.1.2 COS

機　　能　数式のコサインを与えます。

書　　式

```
COS (x)
```

x…コサインを求める数式。単位はラジアンです。

省 略 形　なし

解　　説　数式のコサイン、cos Xが、この関数の値になります。

サンプルプログラム

```
LIST
10 P=PAI(1)
20 PRINT COS(0)
30 PRINT COS(P)
RUN
 1
-1
OK
```

PAIについては、PAI関数参照。

3.1.3 TAN

機　能　数式のタンジェントを与えます。

書　式

```
TAN (x)
```

x…タンジェントを求める数式。単位はラジアンです。

省略形　TA.

解　説　数式のタンジェント、tan Xが、この関数の値になります。

サンプルプログラム

```
LIST
10 A=.5
20 PRINT TAN(.235)
30 PRINT TAN(.235*A)
OK
RUN
 .2394237
 .11804375
OK
```

3.1.4 ATN

機　能　数式のアークタンジェントを与えます。

書　式

```
ATN (x)
```

x…アークタンジェントを求める数式。

省略形　AT.

解　説　数式のアークタンジェント、arctan Xが、この関数の値になります。

サンプルプログラム

```
LIST
10 A=1
20 X=ATN(A)
30 Y=X*4
40 PRINT X
50 PRINT Y
OK
RUN
 .78539816
 3.1415927
OK
```

3.1.5 ABS

機　能
数式の絶対値を与えます。

書　式

```
ABS (x)
```

x …絶対値を求める数式。

省略形
AB.

解　説
数式 x の絶対値｜ x ｜が、この関数の値になります。

サンプルプログラム

```
LIST
10 PRINT ABS(23)
20 PRINT ABS(-23)
30 PRINT ABS(10-33)
OK
RUN
 23
 23
 23
OK
```

3.1.6 SGN

機　能
数式の符号を与えます。

書　式

```
SGN (x)
```

x …数式。

省略形
SG.

解　説
数式 x によって、次の値がこの関数の値になります。

x ＞0のとき　1
x ＝0のとき　0
x ＜0のとき　－1

サンプルプログラム

```
LIST
10 PRINT SGN(93)
20 PRINT SGN(0)
30 PRINT SGN(-93)
OK
RUN
 1
 0
-1
OK
```

3.1.7 INT

機能　数式を超えない最大の整数を与えます。

書式

```
INT (x)
```

x …数式。

省略形　なし

解説　数式を超えない最大の整数[x]が、この関数の値になります。

サンプルプログラム

```
LIST
10 PRINT INT(9.3)
20 PRINT INT(-9.3)
30 PRINT INT(-.5)
OK
RUN
 9
-10
-1
OK
```

3.1.8 FIX

機能　数式の整数部を与えます。

書式

```
FIX (x)
```

x …数式。

省略形　なし

解説　数式の整数部のみを取り出して、小数点以下を切り捨てた値が、この関数の値になります。数値の四捨五入にはCINT(CINT文参照)を使います。

サンプルプログラム

```
LIST
10 A=FIX(2.23)
20 B=FIX(-2.23)
30 C=FIX(A*B)
40 PRINT A,B,C
OK
RUN
 2            -2            -4
OK
```

3.1.9 FRAC

機　能　数式の小数部を与えます。

書　式

```
FRAC（x）
```

x…数式。

省略形　FR.

解　説　数式の小数部のみを取り出して、整数部を取り去った値が、この関数の値になります。

サンプルプログラム

```
LIST
10 A=23.565
20 B=FRAC(A)
30 C=FIX(A)
40 PRINT"A=";A
50 PRINT"B=";B
60 PRINT"C=";C
70 PRINT"B+C=";B+C
OK
RUN
A= 23.565
B= .565
C= 23
B+C= 23.565
OK
```

Aの値を23.565として、

Aの小数部をFRAC関数で求めて、Bに代入し、

Aの整数部をFIX関数で求めて、Cに代入します。

B＋Cを計算すると、もとのAの値になります。

注）整数部の桁数が大きい数のFRACは、誤差が大きくなります。

3.1.10 CINT・CSNG・CDBL

機　能　数式の型変換を行います。

書　式

CINT（x）

CSNG（x）

CDBL（x）

x…型変換を行う数式。

省略形　CIN.
CSN.
CD.

解　説　CINTはxを整数型に、
CSNGはxを単精度型に、
CDBLはxを倍精度型に、
それぞれ変換した値になります。

サンプルプログラム

```
LIST
10 X%=CINT(23.54)
20 Y!=CSNG(1.234567899#)
30 Z#=CDBL(23.54)
40 PRINT X%;Y!;Z#
OK
RUN
 24   1.2345679   23.53999999910593
OK
```

変数や定数の最後に%をつけると整数型
!をつけると単精度型
#をつけると倍精度型
の数値をそれぞれ表します。単精度型の数値の!は省略することができます。
上のサンプルプログラムでは、
23.54を整数の24に、
1.234567899を単精度8桁の1.2345679に、
23.54を倍精度16桁の23.53999999910593に、
それぞれ変換しています。

注）CDBLは単に数値の型を倍精度にするのみで、値そのものの精度は変わりません。値も倍精度にするには、VALとSTR$を用いて以下の様にして下さい。

3.1.11 SQR

機能　数式の平方根を与えます。

書式

```
SQR (x)
```

x…平方根を求める数式。0以上の値です。

省略形　SQ.

解説　数式の平方根$\sqrt{x}$が、この関数の値になります。
0から5までの平方根をSQR関数を使って求めてみましょう。

サンプルプログラム

```
LIST
10 FOR I=0 TO 5
20 PRINT I;SQR(I)
30 NEXT
OK
RUN
 0  0
 1  1
 2  1.4142136
 3  1.7320508
 4  2
 5  2.236068
OK
```

3.1.12 EXP

機能　数式の指数関数を与えます。

書式

```
EXP (x)
```

x…指数関数を求める数式。$x \leqq 88.02969$

省略形　EX.

解説　自然対数の底eのx乗e^{x}が、この関数の値になります。
e15とe0.000002345を求めてみます。

サンプルプログラム

```
LIST
10 PRINT EXP(15)
20 PRINT EXP(.000002345#)
OK
RUN
 3269017.3
 1.00000234500275
OK
```

3.1.13 LOG

機　能
数式の自然対数を与えます。

書　式

```
LOG（x）
```

x…数式。x＞0

省略形
なし

解　説
数式の自然対数ln x(log ex)が、この関数の値になります。log BAはLOG(A)/LOG(B)で求めることができます。ただし、B＞0，B≠1。
公式

$\ln A + \ln B = \ln(AB)$

$\ln A - \ln B = \ln\left(\frac{A}{B}\right)$

を実際に確かめてみましょう。

サンプルプログラム

```
LIST
10 A=2
20 B=5
30 PRINT LOG(A),LOG(B)
40 PRINT LOG(A)+LOG(B),LOG(A*B)
50 PRINT LOG(A)-LOG(B),LOG(A/B)
OK
RUN
 .69314718          1.6094379
 2.3025851          2.3025851
-.91629073         -.91629073
OK
```

3.1.14 PAI

機　能
円周率πのx倍を与えます。

書　式

```
PAI（x）
```

x…数式。

省略形
なし

解　説
円周率πのx倍が、この関数の値になります。
球の表面積を求めるプログラムを作ってみましょう。
　球の表面積Sは、　　　$S = 4\pi r^2$
で求められ、下のサンプルでは、2通りの式を使って計算しています。

サンプルプログラム

```
LIST
10 'キュウ ノ ヒョウメンセキ
20 INPUT"ハンケイ=";R
30 VA=4*π*R*R
40 VB=PAI(4)*R^2
50 PRINT"メンセキ=";VA;VB
OK
RUN
ハンケイ=? 10
メンセキ= 1256.6371  1256.6371
OK
```

注）PAI（1）の代わりに、システム変数のπ（パイ）を使うことができます。
　πは単精度で3.1415927ですが、π#として倍精度型にすると、3.141592653589793として使えます。
　（πは GRAPH を押しながら A を押すと表示されます。）

3.1.15 RAD

機　　能　度単位の数式をラジアン単位に変換します。

書　　式

```
RAD（x）
```

x…ラジアン単位に変換する数式。単位は度です。

省　略　形　なし

解　　説　度単位の数式を$\frac{\pi}{180}$倍して、ラジアン単位に変換します。三角関数の値を求めるときに使うと便利です。

サンプルプログラム　0°，30°，45°，60°，90°のおのおのについてsinとcosの値を求めたプログラムを下に示します。

```
LIST
10 A=RAD(0)
20 B=RAD(30)
30 C=RAD(45)
40 D=RAD(60)
50 E=RAD(90)
60 PRINT USING"##.####";SIN(A);COS(A)
70 PRINT USING"##.####";SIN(B);COS(B)
80 PRINT USING"##.####";SIN(C);COS(C)
90 PRINT USING"##.####";SIN(D);COS(D)
100 PRINT USING"##.####";SIN(E);COS(E)
OK
RUN
 0.0000 1.0000
 0.5000 0.8660
 0.7071 0.7071
 0.8660 0.5000
 1.0000 0.0000
OK
```

3.1.16 FAC

機　　能　数式の階乗を与えます。

書　　式

```
FAC（n）
```

n…階乗を求める数式。0～33の整数。

省 略 形　なし

解　　説　数式の階乗x！が、この関数の値になります。
0から10までの各整数について、階乗計算をしてみましょう。

サンプルプログラム

```
LIST
10 FOR I=0 TO 10
20 PRINT I;FAC(I)
30 NEXT
OK
RUN
 0  1
 1  1
 2  2
 3  6
 4  24
 5  120
 6  720
 7  5040
 8  40320
 9  362880
 10  3628800
OK
```

3.1.17 SUM

機　　能　数式 n の $\sum_{i=1}^{n} i$ を与えます。

書　　式

```
SUM (n)
```

n …数式。0以上の整数値です。

省 略 形　なし

解　　説　$\sum_{i=1}^{n} i = 1+2+\cdots+(n-1)+n = \frac{1}{2}n(n+1)$ が、この関数の値になります。

サンプルプログラム

```
LIST
10 FOR I=0 TO 10
20 PRINT I;SUM(I)
30 NEXT
OK
RUN
 0   0
 1   1
 2   3
 3   6
 4   10
 5   15
 6   21
 7   28
 8   36
 9   45
 10   55
OK
```

3.1.18 RND

機　　能　0以上1未満の乱数を発生します。

書　　式

```
RND [ (x) ]
```

x …数式。

省 略 形　RN.

解　　説　発生した擬似乱数が、この関数の値になります。この乱数は、0以上1未満の乱数になっています。

サンプルプログラム　乱数を10個発生させてみましょう。

```
LIST
10 FOR I=1 TO 10
20 PRINT I;RND(1)
30 NEXT
OK
RUN
 1  .6475668
 2  .58308077
 3  .17951131
 4  .14283574
 5  .13336551
 6  .60755396
 7  .48844886
 8  .68064535
 9  .71988237
 10  .13614893
OK
```

3.2 文字関数

3.2.1 ASC

機　能　文字コードを数値に変換します。（⇐⇒ CHR$ ）

書　式

ASC（文字列）

文字列……数値に変換する文字が左端にある文字列。

省略形　なし。

解　説　文字列の最初の1文字のキャラクタコードが、この関数の値になります。キャラクタコードの値は0～255の整数値です。（巻末「キャラクタコード表」参照）文字列は式や変数でかまいません。また、文字列がヌルストリングのときに、この関数の値は0になります。

サンプルプログラム　"A","B","C" と ABC のキャラクタコードに変換します。"ABC" のときは最初の "A" のみコードに変換します。

```
LIST
10 A$="A":B$="B":C$="C":D$="ABC"
20 X=ASC(A$):Y=ASC(B$):Z=ASC(C$):Q=ASC(D$)
30 PRINT X;Y;Z;Q
OK
RUN
 65  66  67  65
OK
```

3.2.2 CHR$

機　能　数値をキャラクタコードとみなして、対応する文字に変換します。（⇐⇒ASC）

書　式

CHR$（$x_1$ [, x_2 , x_3 ,……]）

x_1 , x_2 , x_3 ,……: 文字に変換する数式。
各0～255

省略形　CHR.

解　説　数値をキャラクタコードとする文字が与えられます。数値1つにつき1文字が得られますが、数値を1つずつ ,（カンマ）で区切って入れると、その数値の並びに対応する文字列が得られます。
キャラクタコードについては巻末の「キャラクタコード表」を参照してください。

サンプルプログラム　キャラクタコードを文字に変換します。

65→A
60→B
70→L，79→O，86→V，69→E，76→L，89→Y，33→！
と各コードに対応する文字が得られます。

```
LIST
10 X$=CHR$(65)
20 Y$=CHR$(66)
30 Z$=CHR$(76,79,86,69,76,89,33)
40 PRINT X$,Y$,Z$
OK
RUN
A          B          LOVELY!
OK
W. 1:LINP.A$:LP.A$:WE.
```

3.2.3 VAL

機能 数字の文字列を数値に変換します。

書式

VAL（文字列）

文字列……数字の文字列。

省略形 VA.

解説 文字列の中の文字としての数字を数値に変換します。もし文字列の最初の文字が＋，－，＆または数字でないときには、この関数の値は0になります。また、文字列の中に数字以外の文字（16進数の場合はA～Fを除く）が現れたら、それ以降の文字は無視します。
この関数は常に倍精度の値をもちます。

サンプルプログラム

```
LIST
10 A=VAL("2356")
20 B=VAL("&H"+"EFC0")
30 C=VAL("&O"+"177666")
40 D=VAL("&B"+"101000011011")
50 A$=STR$(A)
60 B$=HEX$(B)
70 C$=OCT$(C)
80 D$=BIN$(D)
90 PRINT A,A$
100 PRINT B,B$
110 PRINT C,C$
120 PRINT D,D$
OK
RUN
 2356         2356
-4160        EFC0
-74          177666
 2587        101000011011
OK
```

STR$, HEX$, OCT$, BIN$についてはその関数の項を参照してください。

3.2.4 STR$

機能 数式の値を文字の数字に変換します。

書式

STR $ (x)

x……数式。

省略形 なし。

解説 数式および数値を表す文字列が、この関数の値になります。
正の数の場合。先頭にスペースが一文字はいります。

サンプルプログラム

```
LIST
10 PRINT STR$(0)
20 PRINT STR$(100)
30 PRINT STR$(255)
OK
RUN
 0
 100
 255
OK
```

3.2.5 HEX$

機能　数式を16進数の文字列に変換します。

書式

```
HEX$ (x)
```

x……数式。 -32768～65535

省略形　HE.

解説　数式を16進数の文字列に変換したものが、この関数の値になります。

サンプルプログラム

```
LIST
10 PRINT HEX$(-32768!)
20 PRINT HEX$(0)
30 PRINT HEX$(235)
40 PRINT HEX$(65535!)
OK
RUN
8000
0
EB
FFFF
OK
```

3.2.6 OCT$

機能　数式を8進数の文字列に変換します。

書式

```
OCT$ (x)
```

x……数式。 -32768～65535。

省略形　OC.

解説　数式を8進数の文字列に変換したものが、この関数の値になります。

サンプルプログラム

```
LIST
10 PRINT OCT$(-32768!)
20 PRINT OCT$(0)
30 PRINT OCT$(235)
40 PRINT OCT$(65535!)
OK
RUN
100000
0
353
177777
OK
```

3.2.7 BIN$

機能 数式を2進数の文字列に変換します。

書式

```
BIN$ (x)
```

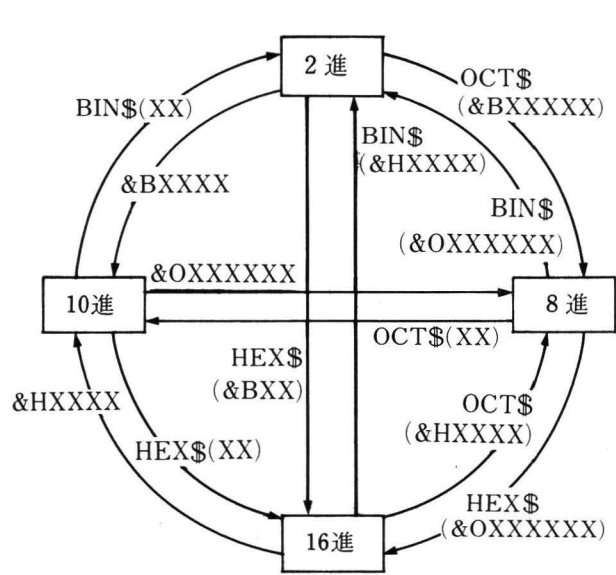
x……数式。 -32768～65535

省略形 BI.

解説 数式および数値を2進数の文字列（0, 1からなる）に変換したものが、この関数の値になります。

サンプルプログラム

```
LIST
10 PRINT BIN$(-32768!)
20 PRINT BIN$(0)
30 PRINT BIN$(235)
40 PRINT BIN$(65535!)
OK
RUN
1000000000000000
0
11101011
1111111111111111
OK
```

〈HEX$, OCT$, BIN$ および定数について〉

10進	2進	8進	16進	
0	00000	00	00	
1	00001	01	01	
2	00010	02	02	
3	00011	03	03	
4	00100	04	04	
5	00101	05	05	
6	00110	06	06	
7	00111	07	07	
8	01000	10	08	
9	01001	11	09	
10	01010	12	0A	
11	01011	13	0B	
12	01100	14	0C	
13	01101	15	0D	
14	01110	16	0E	
15	01111	17	0F	
16	10000	20	10	
17	10001	21	11	
18	10010	22	12	
100	1100100	144	64	
1000	1111101000	1750	3E8	
-1	1111111111111111	177777	FFFF	補数表現
-2	1111111111111110	177776	FFFE	補数表現
-10	1111111111110110	177766	FFF6	補数表現
-100	1111111110011100	177634	FF9C	補数表現
-1000	1111110000011000	176030	FC18	補数表現

3.2.8 LEFT$

機能　文字列の左側から、指定した数だけ文字を取り出します。(⇐⇒RIGHT$)

書式

```
LEFT$ (文字列 , n )
```

n…文字列から取り出す文字数。0～255

省略形　LEF.

解説　文字列の左側から取り出されたn個の文字列が、この関数の値になります。nの値が文字列の文字数より大きいときには、すべての文字が、n＝0のときには、ヌルストリングがそれぞれ関数の値になります。

サンプルプログラム

```
LIST
10 A$="ABCDEFGHI"
20 PRINT LEFT$(A$,10)
30 PRINT LEFT$(A$,4)
40 PRINT LEFT$(A$,0)
OK
RUN
ABCDEFGHI
ABCD

OK
```

3.2.9 RIGHT$

機能　文字列の右側から、指定した数だけ文字を取り出します。(⇐⇒LEFT$)

書式

```
RIGHT$ (文字列 , n )
```

n…文字数。0～255

省略形　RI.

解説　文字列の右側から取り出されたn個の文字列が、この関数の値になります。nの値が文字列の文字数より大きいときには、すべての文字が、n＝0のときには、ヌルストリングがそれぞれ関数の値になります。

サンプルプログラム

```
LIST
10 A$="ABCDEFGHI"
20 PRINT RIGHT$(A$,10)
30 PRINT RIGHT$(A$,0)
40 PRINT RIGHT$(A$,4)
OK
RUN
ABCDEFGHI

FGHI
OK
```

3.2.10 MID$

機　　能　文字列の中から、指定した数だけ一連の文字を取り出します。(⇒LEFT$、RIGHT$)

書　　式

```
MID$ (文字列 , 開始位置[, n ])
```

開始位置…文字列の最初の文字の位置を1とする文字の位置。

n …………文字列から取り出す文字の数。0 ～255

省 略 形　MI.

解　　説　文字列の中の開始位置から取り出されたn個の文字列が、この関数の値になります。nを省略したとき、および文字列の文字数よりnが大きいときには、開始位置から右端までの文字列が、関数の値になります。開始位置が文字列の数より大きいときには、ヌルストリングが関数の値になります。

サンプルプログラム

```
LIST
10 A$="ABCDEFGHI"
20 PRINT MID$(A$,3,5)
30 PRINT MID$(A$,8)
40 PRINT MID$(A$,3,20)
OK
RUN
CDEFG
HI
CDEFGHI
OK
```

3.2.11 STRING$

機　　能　任意の文字列を任意の数だけ用意します。

書　　式

```
STRING$ (n, { 文字列 })
            { 数  式 }
```

n ………文字列の数。0 ～255

数式……0 ～255のアスキーコードとみなされます。

省 略 形　STRIN.

解　　説　文字列または数式で指定した文字列をn個並べて用意します。

サンプルプログラム

```
LIST
10 A$="ABCDE"
20 PRINT STRING$(3,A$)
OK
RUN
ABCDEABCDEABCDE
OK
```

STRING$ (3, A$)は、A$＋A$＋A$と同じです。

3.2.12 SPACE$

機能　スペースからなる文字列を与えます。

書式

```
SPACE$（n）
```

n…スペースの数。0～255

省略形　SPA.

解説　n個のスペース(空白)からなる文字列が、この関数の値になります。

サンプルプログラム

```
LIST
10 PRINT "A";SPACE$(10);"A"
OK
RUN
A          A
OK
```

3.2.13 INSTR

機能　文字列の中に含まれる特定の文字列の位置を検出します。

書式

```
INSTR（[開始位置 ,] 文字列1, 文字列2）
```

開始位置…探し始める文字の位置。1～255
　　　　　文字列1の最初の文字の位置を1とします。

省略形　INS.

解説　文字列1の中を、開始位置から順に右へ、文字列2があるかどうか探して行きます。もしみつかれば、その先頭の文字の位置を関数の値とし、みつからなければ、関数の値を0にします。開始位置を省略すると、文字列1の最初から探し出します。

サンプルプログラム

```
LIST
10 A$="SHARP HuBASIC"
20 PRINT INSTR(A$,"Hu")
OK
RUN
 7
OK
```

3.2.14 LEN

機能　文字列の文字数を与えます。

書式

```
LEN（文字列）
```

省略形　LE.

解説　文字列に含まれるすべての文字の数が、この関数の値になります。文字の数は0～255で、文字列がヌルストリングのときは0になります。なお、空白やコントロールコードなどの画面表示されない文字も1文字として数えます。

サンプルプログラム

```
LIST
10 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 PRINT LEN(A$)
OK
RUN
 26
OK
```

アルファベットの文字列の長さは26文字です。

3.2.15 HEXCHR$

機能　16進数の文字列を、文字列に変換します。

書式
```
HEXCHR$ (文字列)
```

文字列…16進数の文字列

省略形　HEXC.

解説　16進数の文字列を最初から2文字(2バイト)ずつ区切って、その2文字に対応するキャラクタに変換して文字列を作ります。この文字列が、この関数の値になります。16進数の文字列の中にスペース(空白)があるときは、区切れとみなし、無視して次へ進みます。

サンプルプログラム
```
LIST
10 A$="1234567890ABCD"
20 PRINT HEXCHR$(A$)
OK
RUN
4Vxーオヘ
OK
```

"34567890ABCD"を、
"34 56 78 90 AB CD" と2文字づつ区切ってそれに対応する文字列
"4 V X ー オ ヘ" を作ります。

3.2.16 MIRROR$

機能　文字列のバイナリーコードを転置した文字列を与えます。

書式
```
MIRROR (文字列)
```

省略形　MIR.

解説　文字列のバイナリーコード(2進コード)を左右転置したコードに対応する文字列が、この関数の値になります。

サンプルプログラム
```
LIST
10 A$=CHR$(&H83)
20 PRINT BIN$(ASC(A$)),BIN$(ASC(MIRROR$(A$)))
OK
RUN
10000011  11000001
OK
```

ユーザー定義のキャラクタなどのパターンを左右にひっくり返すときなど便利です。
(DEF CHR$参照)

3.2.17 MKI$・MKS$・MKD$

機　　能

数式をそのまま内部表現(バイナリィコード)に対応した文字列に変換します。
(⇔CVI、CVS、CVD)

書　　式

MKI$ (整数値)

MKS$ (単精度値)

MKD$ (倍精度値)

省 略 形

MK.
MKS.
MKD.

解　　説

MKI$は整数値を2バイトの文字列に、
MKS$は単精度値を5バイトの文字列に、
MKD$は倍精度値を8バイトの文字列に、
それぞれ変換します。
ファイルに対して数字を記録する場合に用いると、ファイルのスペースを節約できます。

サンプルプログラム

```
LIST
10 A%=-32:B!=3.141592:C#=1.234567890123456#
20 A$=MKI$(A%)
30 B$=MKS$(B!)
40 C$=MKD$(C#)
50 X%=CVI(A$)
60 Y!=CVS(B$)
70 Z#=CVD(C$)
80 PRINT A%;B!;C#
90 PRINT A$:PRINT B$:PRINT C$
100 PRINT X%;Y!;Z#
OK
RUN
-32  3.141592  1.234567890123456
●テ
  ラ◆
Rｂマチ
-32  3.141592  1.234567890123456
OK
```

3.2.18 CVI・CVS・CVD

機　　能　文字列を数値に変換します。(⟺MKI$、MKS$、MKD$)

書　　式

CVI（2文字の文字列）

CVS（5文字の文字列）

CVD（8文字の文字列）

省 略 形　CVI、CVDにはなし。
CVSの省略形はCV.です。

解　　説　CVIは2文字(2バイト)の文字列を整数値に、
CVSは5文字(5バイト)の文字列を単精度実数値に、
CVDは8文字(8バイト)の文字列を倍精度実数値に、
それぞれ変換します。
ファイルから読み込んだ文字データを数値に変換する場合に、これらの関数を使うと便利です。

サンプルプログラム

```
LIST
10 A$="AB":B$="ABCDE":C$="ABCDEFGH"
20 A%=CVI(A$)
30 B!=CVS(B$)
40 C#=CVD(C$)
50 X$=MKI$(A%)
60 Y$=MKS$(B!)
70 Z$=MKD$(C#)
80 PRINT A$:PRINT B$:PRINT C$
90 PRINT A%;B!;C#
100 PRINT X$:PRINTY$:PRINTZ$
OK
RUN
AB
ABCDE
ABCDEFGH
 16961  8.2273479E-20  8.227347927418453D-20
AB
ABCDE
ABCDEFGH
OK
```

3.3 特殊関数

3.3.1 INP

機能 I/Oアドレスの I/Oポートから1バイトのデータを読み込みます。

書式

INP (I/Oアドレス)

I/Oアドレス…読み込むI/Oアドレス。 -32768〜65535

省略形 なし

解説 I/Oアドレスで指定したI/Oポートから読み込んだ1バイトのデータが、この関数の値になります。値は0〜255です。

画面のホーム位置(左上隅)にカーソルをセットして、「AB」と打ち込んでから、次のプログラムを実行してみてください。

サンプルプログラム

```
AB
LIST
10 X=INP(&H3000)
20 Y=INP(&H3001)
30 PRINT X,Y
40 PRINT CHR$(X);SPC(9);CHR$(Y)
OK
RUN
 65          66
A            B
OK
```

画面(テキスト画面)は、I/Oポートの&H3000〜&H37FFに割りつけられています。

3.3.2 PEEK

機能 指定したメモリーアドレスからデータを取り出します。(⟺POKE)

書式

```
PEEK (アドレス)
```

アドレス…メインメモリー内のアドレス。0～&HFFFF

省略形 PE.

解説 メインメモリー内の指定したアドレスから取り出した1バイトのデータが、この関数の値になります。
PEEK関数を使ってメモリーダンププログラムを作ってみましょう。
INPUT文で、読み込み開始アドレスと終了アドレスを指定すると、その領域内からデータを読み込んでメモリーのダンプ(dump)をします。
桁をそろえて表示するために、RIGHT$関数を使用しています。

サンプルプログラム

```
LIST
10 INPUT"Start Address=&H",SA$
20 INPUT"End   Address=&H",EA$
30 SA=VAL("&H"+SA$)
40 EA=VAL("&H"+EA$)
100 PRINT RIGHT$("000"+HEX$(SA),4);"   ";
110 FOR I=SA TO SA+7
120  DT=PEEK(I)
130  PRINT RIGHT$("0"+HEX$(DT),2);" ";
140 NEXT
150 SA=SA+8
160 PRINT
170 IF SA<=EA THEN 100
OK
RUN
Start Address=&H5000
End   Address=&H501F
5000  C8 2C 7D E6 07 E5 21 00
5008  08 20 EB 3A 07 00 6F 26
5010  C8 18 E3 E5 21 00 00 B7
5018  ED 52 E1 37 C8 CB 7A C0
OK
```

3.3.3 PEEK@

機　能　VRAM内の指定されたアドレスの内容を読み出します。(⇒INP)

書　式

```
PEEK@（アドレス）
```

アドレス…VRAM内のアドレスで&H2000～&HFFFF。

省略形　PE.@

解　説　VRAM(Video Random Access Memory)の指定されたアドレスの内容1バイトが、この関数の値になります。値は、0～255の整数です。
カーソルをホーム位置(画面左上隅)にもっていって、次のようにキーボードから入力してください。
ホーム位置のアドレスは&H3000、その属性のアドレスは&H2000番地なので、?のキャラクタコードと属性のコードが得られます。(付録「テキスト画面とその属性ポートへのアクセス方法」参照)

サンプルプログラム

```
? HEX$(PEEK@(&H3000))
3F
OK
? PEEK@(&H2000)
 7
OK
```

3.3.4 POS

機　能　画面のカーソルの水平位置を与える。

書　式

```
POS（数式）
```

数式…ダミーの値0

省略形　なし

解　説　画面上の現在のカーソルの水平位置の値(WIDTH40のとき0～39、WIDTH80のとき0～79)が、この関数の値になります。

サンプルプログラム

```
LIST
10 CLS4:LOCATE 12,15
20 X=POS(0)
30 LOCATE 0,0
40 PRINT X
RUN
 12
OK
```

3.3.5 LPOS

機　　能	プリンターヘッドの現在位置を与えます。

書　　式

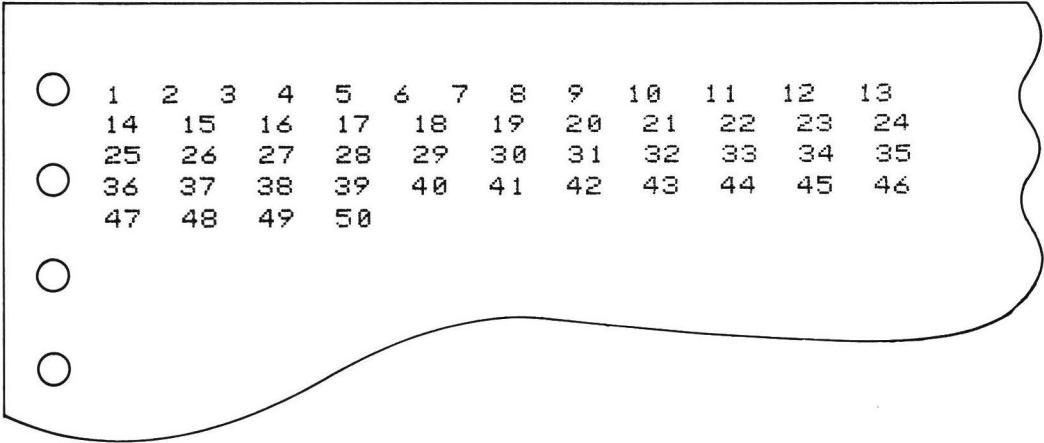
```
LPOS (数式)
```

数式は何を指定してもかまいません。

省 略 形 LPO.

解　　説 プリンターのヘッドの現在位置が、この関数の値になります。値の範囲は0～240です。この関数をうまく使うと、プリントの簡単な制御を行うことができます。

サンプルプログラム プリンターヘッドが40桁を越えると改行するプログラムを下に示します。

```
10 FOR I=1 TO 50
20 IF LPOS(0)>40 THEN LPRINT
30 LPRINT I;
40 NEXT
50 LPRINT
RUN
OK
```

LPRINTは、バッファに1行分のデータがたまると出力するようになっているので、50行目にLPRINTを入れてプリントの最終行のデータ(47～50の4つ)をバッファから吐き出すようにしています。

3.3.6 VARPTR

機　　能 変数が格納されているメインメモリーのアドレスを与えます。

書　　式

```
VARPTR (変数)
```

変数…整数型、単精度型、倍精度型、文字型

省 略 形 VAR.

解　　説 変数の内容を記憶しているメインメモリーの先頭アドレスが、この関数の値になります。(STRPTR参照)

サンプルプログラム

```
A=213:AD=VARPTR(A):PRINT AD;HEX$(AD)
-24902 9EBA
OK
```

変数Aの内容が格納されているアドレスは、9E4E番地であることを示しています。

注) BASICのバージョン及び変数の状態、プログラムの有無によって値が違うことがあります。

3.3.7 FRE・SIZE

機　能　ユーザーメモリーの未使用領域のサイズを与えます。

書　式

```
FRE（数式）又はSIZE
```

数式は何を指定してもかまいません。

省略形　なし

解　説　BASICのプログラムで使っていないユーザーメモリーのバイト数が、この関数の値になります。

サンプルプログラム

```
? FRE(0)
 23536
OK
? SIZE
 23536
OK
```

注）BASICのバージョン，変数の状態及びプログラムの有無によって値が違ってきます。

3.3.8 POINT

機　能　画面の点(ドット)のパレットコードを与えます。

書　式

```
POINT（x, y）
```

x, y…画面上の論理座標(実数値)

省略形　POI.

解　説　画面上の1ドットのパレットコード(0～7)を与えます。パレットコードについてはPALET文を参照してください。WINDOW外の点は-1の値を与えます。

サンプルプログラム

```
IF POINT(X,Y)=2 THEN 100
```

上の文は「画面上の（X,Y）のドットがパレットコードの2だったら行番号100へジャンプせよ。」という意味です。

3.3.9 STICK

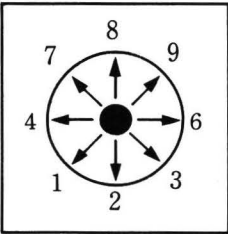
機　能　ジョイスティックまたはテンキーからの入力値を与えます。

書　式

```
STICK (x)
```

x……0, 1, 2
　　0…テンキー
　　1…ジョイスティック1
　　2…ジョイスティック2

省略形　STI.

解　説　ジョイスティックまたはキーボードのテンキーから入力された値が、この関数の値になります。
ジョイスティックに接続している場合には、スティックを倒す方向によって、下の図のような値が入力されます。

キーボード上のテンキーからは1～9の値のみ入力され、ほかのキーが押されたときには、0が入力されます。フルキーの中の1～9のキーでも1～9の値が入力されます。

サンプルプログラム

```
LIST
10 PRINT STICK(0);:GOTO10
OK.
```

RUNすると画面に連続的に0が表示されますから、テンキーから数字キーを入力して、入力値が正しいかどうか確めてください。
止めるときは、SHIFT と BREAK キーを同時に押すか、CTRL と C キーを同時に押してください。「Break in 10」と表示して止まります。

3.3.10 STRIG

機　能　ジョイスティックのトリガーボタンまたはキーボードのスペースキーの状態を与えます。

書　式

```
STRIG (x)
```

x……0、1、2
　　0…キーボードのスペースキー
　　1…ジョイスティック1 のボタン
　　2…ジョイスティック2 〃

省略形　STR.

解　説　ジョイスティックの(トリガー)ボ タンまたはキーボード上のスペースキーが、
押されているときには真の値 -1 が、
押されていないときには偽の値0が、
この関数の値になります。

サンプルプログラム

```
IF STRIG(0) GOTO "BEAM"
```

ジョイスティックのボタンが押されていればラベル名の"BEAM"にジャンプさせる文です。

3.3.11 CMT

機能　カセットの状態を与えます。

書式

```
CMT［（x）］
```

x……0,1,2

省略形　CM.

解説　（1）CMTのみで引数が省略されている場合、

カセットの状態	関数CMTの値
EJECT	0
STOP	1
PLAY	2
FAST	3
REW	4
RECODE	10

（2）CMT（x）の書式の場合、

xの値	カセットの状態	関数CMT（x）の値
0	カセットテープの回転しているとき	－1
	カセットテープの回転が止まっているとき	0
1	カセットテープがセットされているとき	－1
	カセットテープがセットされていないとき	0
2	カセットテープの消去防止用のツメが折ってあるとき	0
	カセットテープの消去防止用のツメを折っていないとき	－1

サンプルプログラム

```
IF CMT=0 THEN PRINT "フタヲ シメロ"
```

上の文は、「もしカセットレコーダーのフタが開いていたら画面に『フタヲシメロ』と表示せよ」という意味です。

3.3.12 EOF

機能　シーケンシャルファイルの終了を検出します。

書式

```
EOF（ファイル番号）
```

ファイル番号…OPEN（OPEN文参照）で指定したファイル番号。

省略形　EO.

解説　ファイル番号で指定したファイルからデータを読み込んだときに、
ファイルの最後のデータであれば真の値－1が、
ファイルの最後のデータでなければ偽の値0が、
この関数の値になります。

サンプルプログラム

```
IF EOF(1) THEN CLOSE#1
```

上の文は「ファイル番号1のファイルが終りのデータならばファイルを閉じる。」という意味をもっています。

3.3.13 KANJI$

機　能　漢字ROMから読み込んだ漢字パターンデータが、この関数の値になります。

書　式

```
KANJI$（x）
```

x…漢字コード。整数。

省略形　KA.

解　説　専用の漢字ROMから読み込んだ漢字パターン(縦16、横16ドットのサイズ)のコードが、この関数の値になります。

KANJI$は、PATTERN文と一緒に使います。(PATTERN文参照)

サンプルプログラム

```
LIST
10 INIT
20 POSITION 0,0
30 PATTERN -16,KANJI$(&H7F1)
OK
RUN
OK
```

このサンプルがKANJI$の基本的な使い方を示しています。

実際に実行すると画面左上隅に漢字の「漢」を表示します。(PATTERN文参照)

専用の漢字ROMボードがI/Oポートにそう入されてない場合は白い正方形が表示されます。

3.3.14 FN

機　能　関数名と引数を与えて、関数ルーチンのコールを行います。

書　式

```
FN関数名[（引数1[,引数2]…）]
```

関数名…ユーザーの定義する関数名。文字型でもよい。

引数……この引数をDEFFNで定義した関数の引数の値とする。

省略形　なし

解　説　DEF FN参照。

サンプルプログラム

```
LIST
10 DEF FNA$(X)=RIGHT$("000"+HEX$(X),4)
20 PRINT FNA$(10)
30 PRINT FNA$(1000)
40 PRINT FNA$(10000)
OK
RUN
000A
03E8
2710
OK
```

3.3.15 USR

機　能　引数を与えて、機械語サブルーチンのコールを行います。

書　式

```
USR［ユーザー関数番号］（引数）
```

ユーザー関数番号…DEF USRで定義した番号　0～9

引数……………………ユーザー関数に渡すデータ

省略形　なし

解　説　引数を機械語サブルーチン(ユーザー関数)に渡して、サブルーチンのコールを行います。引数を持たすことができるので、CALL(CALL文参照)とは区別します。

この関数の詳細は、「ユーザー命令の使い方」を参照してください。

サンプルプログラム

```
Z=USR1(10)
```

3.4 入出力用文字関数

3.4.1 MEM$

機　能　指定したメモリーから任意の数だけ文字を取り出します。

書　式
```
MEM$ (アドレス , n)
```

アドレス…メインメモリー内のアドレス。0～&HFFFF

n …………文字の数。0～255

省略形　MEM.

解　説　メインメモリー内の指定したアドレスから始まる領域から、n 個の文字を取り出して、関数の値にします。

サンプルプログラム
```
M$=MEM$(&HFE00,255)
```

3.4.2 SCRN$

機　能　テキスト画面上の文字列を与えます。

書　式
```
SCRN$ (x, y, n)
```

x, y…カーソルの絶対座標。x が水平位置、y が垂直位置

n …………0～255

省略形　SCRN.

解　説　テキスト画面上の水平座標 x 、垂直座標 y から n 個の文字が、この関数の値になります。

n が 0 のとき、関数の値はヌルストリングになります。

カーソルをホーム位置にして10数個、文字を打ち込んで、次のプログラムを実行してみてください。(ただし、プログラムは画面の下のほうに作ること) 注) ホーム位置は画面の左上隅です。

サンプルプログラム
```
ABCDEFGHIJKLMNOPQRSTU


PRINT SCRN$(0,0,10)
ABCDEFGHIJ
OK
```

3.4.3 CHARACTER$

機　能　指定したカーソル位置の文字を与えます。　(⇒SCRN$)

書　式
```
CHARACTER$ (x, y)
```

x, y……カーソルの座標 x 列 y 行

x …水平位置　WIDTH 40のとき　0～39

　　　　　　　WIDTH 80のとき　0～79

y …垂直位置　0～24

省略形　CHAR.

解　説　x 列 y 行のカーソル位置にある文字が、この関数の値になります。これはSCRN$ (x, y, 1) と同じものと考えることができます。

サンプルプログラム
```
LIST
10 LOCATE 0,0:PRINT"*"
20 A$=CHARACTER$(0,0)
30 LOCATE 0,10:PRINTA$
OK
RUN

*
OK
```

3.4.4 CGPAT$

機　　能　キャラクタコードに対応する32バイト(32文字)のストリングを与えます。

書　　式

```
CGPAT$ (キャラクタコード)
```

キャラクタコード…0～255

省 略 形　CGP.

解　　説　キャラクタコードとユーザー定義のキャラクタに対応する32バイト(32文字)のストリングデータを返します。左から8文字がノーマルキャラクタのビットパターンで作られた文字、次の8文字がDEFCHR$文によって、ユーザーが定義したキャラクタの青色の部分のビットパターンで作られた文字、次の8文字が赤色の部分で、最後の8文字が緑色の部分で作られた文字になります。
たとえば、キャラクタHのコードは&H48ですが、このコードにDEFCHR$によってシアン色の斜線を定義した場合を例にとります。

キャラクタH(キャラクタコード&H48)のドットパターンを下の図に示します。

また、キャラクタコード&H48に下の図のようなキャラクタを定義します。

青と緑の部分　　　　　　赤の部分

サンプル
プログラム

上の例についてのサンプルプログラムを下に示します。

```
LIST
10 SCREEN 0,0,0:WIDTH 40
20 DIM ST$(32)
30 PRINTCHR$(&H48)'ﾓｼﾞ  H ﾉ ﾋｮｳｼﾞ｡
40 '
50 ' ｷｬﾗｸﾀ ﾉ ﾃｲｷﾞ｡
60 A$=HEXCHR$("0102040810204080")
70 N$=HEXCHR$("0000000000000000")
80 DEF CHR$(&H48)=A$+N$+A$
90 '
100 'ｷｬﾗｸﾀ ﾉ ﾋｮｳｼﾞ｡
110 CGEN 1
120 PRINT CHR$(&H48)
130 CGEN 0
140 '
150 'ﾃｲｷﾞｼﾀ ｷｬﾗｸﾀ ﾆ ﾀｲｽﾙ 32ﾓｼﾞ ｦ ｴﾙ｡
160 CG$=CGPAT$(&H48)
170 PRINT#0,CG$
180 '
190 '32ﾓｼﾞ ﾉ ｷｬﾗｸﾀｺｰﾄﾞ ﾉ ﾋｮｳｼﾞ｡
200 FOR I=1 TO 32
210 ST$(I)=MID$(CG$,I,1)
220 PRINT USING"####";ASC(ST$(I));
230 IF (I MOD 8)=0 THEN PRINT
240 NEXT
OK
```

170行目のPRINT＃0は、CGPAT$(＆H48)に含まれているコントロールキャラクタを表示するために使っています。
参考のため、200～240行でCGPAT$(＆H48)の各キャラクタのコードを求めて表示しています。
下が実行させた結果で、3行目がCGPAT$(＆H48)の32文字の文字列です。2行目の／は、実際にはシアン色で表示されます。

```
H
/
BBB‾BBB^@^A^B^D^H^P @_^@^@^@^@^@^@^@^@^A^B^D^H^P @_
  66  66  66 126  66  66  66   0
   1   2   4   8  16  32  64 128
   0   0   0   0   0   0   0   0
   1   2   4   8  16  32  64 128
Ok
```

4行目は、3行目のBBB‾BBB^@の各キャラクタを10進コードに変換したもので、これの2進コードがHのドットパターンを表しています。
5行目は、3行目の^A^B^D^H^P　@　の各キャラクタをコードに変換したもので、これの2進コードが2行目の／の青の部分のドットパターンを表しています。
6行目は、^@^@^@^@^@^@^@^@の各キャラクタをコードに変換したもので、／の赤の部分のドットパターンを表しますが、例ではドットがないので、0が並んでいます。
7行目は、5行目と同じで、この2進コードが／の緑の部分のドットパターンを表しています。

3.4.5 INKEY$

機　能　キーボードからの1文字を与えます。

書　式

```
INKEY$ {(n)
        省略}
```

n＝0、1、2

省略形　INK.

解　説　キーボードから入力された1文字が、この関数の値になります。

引数の状態によって、4つの機能をもっています。

（1）引数が省略された場合

キーが押された時、その文字が関数の値になり、

押されていなければ、ヌルストリングが関数の値になります。

（2）n＝0の場合

キーが押されていれば、間断なくその文字が関数の値になり、

押されていなければ、ヌルストリングが関数の値になります。

（3）n＝1の場合

カーソルをブリンクし、1文字の入力があるまで待ちます。1文字入力のINPUT$と考えることができます。

（4）n＝2の場合

キーボードからの入力状態を示すサブCPUからの1バイトの情報がこの関数の値になります。。

入力状態を表す情報のビット構成は、次のようになっています。

7	6	5	4	3	2	1	0
T	P	R	G	L	K	S	C

ビット	入力状態	入力有	入力無
T	テンキー部分	0	1
P	キーの入力	0	1
R	リピート機能	働いている0	働いていない1
G	GRAPHキー	0	1
L	CAP LOCKキー	0	1
K	カナモードキー	0	1
S	SHIFTキー	0	1
C	CTRLキー	0	1

サンプル
プログラム

```
IF INKEY$="8" GOTO "UP"
```

3.4.6 INPUT$

機能　キーボードなどの入力装置またはファイルから文字列を読み込みます。

書式

```
INPUT$（n[,[#]ファイル番号]）
```

n………………読み込む文字数　0～255

ファイル番号…OPEN（OPEN文参照）で指定したファイル番号

省略形　I.$

解説　キーボードなどの入力装置またはテープなどのファイルから読み込んだn個の文字が、この関数の値になります。

,[#]ファイル番号を省略してINPUT$（n）とすると、キーボードからn個の文字を読み込みます。なお、BREAKコード（03）を除くすべての文字の読み込みが可能です。

サンプルプログラム

```
IF INPUT$(5)<>"SHARP" THEN PRINT"COMPUTER"
```

3.5 タイマー変数

3.5.1 TIME$

機能 内蔵時計の時間を与えます。

書式

```
TIME$
```

引数はありません。

省略形 なし

解説 内蔵時計の値が、この変数の値になります。

hh：mm：ssの文字式で表されます。

hh …00～23
mm…00～59
ss …00～59

サンプルプログラム

```
PRINT TIME$
13:08:57
OK
```

3.5.2 DAY$

機能 内蔵カレンダーの曜日を与えます。

書式

```
DAY$
```

引数はありません。

省略形 DAY.

解説 内蔵カレンダーの曜日が、この変数の値になります。

SUN.	日曜日
MON.	月曜日
TUE.	火曜日
WED.	水曜日
THU	木曜日
FRI.	金曜日
SAT.	土曜日

サンプルプログラム

```
PRINT DAY$
SUN
OK
```

3.5.3 DATE$

機　　能　内蔵カレンダーの年月日を与えます。

書　　式

```
DATE$
```

引数はありません。

省　略　形　DATE.

解　　説　内蔵カレンダーの年月日が、この変数の値になります。年は初期状態で82年となり、DATE$の代入により変更を受けますが、時間がたっても変化しません。

サンプルプログラム

```
PRINT DATE$
80/01/01
OK
```

3.5.4 TIME

機　　能　内蔵時計によるストップウォッチの時刻を与えます。

書　　式

```
TIME
```

引数はありません。

省　略　形　TIM.

解　　説　TIMEに代入した時刻（0～86399秒までの値）に、経過秒数を加えた値が、この変数の値になります。86399秒を越えると再び0秒に戻ります。
これにより、内蔵時計に影響を与えずに時間を計測することが可能となります。

サンプルプログラム

```
LIST
10 TIME=0
20 FOR I=0 TO 10000:NEXT
30 PRINT TIME
OK
RUN
 6
OK


LIST
10 TIME=0
20 FOR I%=0 TO 10000:NEXT
30 PRINT TIME
OK
RUN
 4
OK
```

3.6 システム変数

3.6.1 CSRLIN

機　能　カーソルの垂直位置を与えます。

書　式

```
CSRLIN
```

引数はありません。

省略形　CSR.

解　説　現在のカーソルの垂直位置が、この変数の値になります。値の範囲は0～24です。

サンプルプログラム

```
LOCATE 5,10:PRINT POS(0);CSRLIN



          5   10
OK
```

3.6.2 STRPTR

機　能　文字データが格納されているメインメモリー内の領域の先頭アドレスを与えます。(⇒VARPTR)

書　式

```
STRPTR
```

引数はありません。

省略形　STRP.

解　説　VARPTR(VARPTR文参照)では、整数型、単精度型、倍精度型の数値データについては、そのデータの格納されているメモリーの先頭アドレスが与えられますが、文字データについては、その文字列の長さと、文字データの格納されている領域の先頭アドレスを0とする相対アドレスが与えられます。よって、文字データの格納されている実際のアドレスを知るためには、文字データの格納されている領域の先頭アドレスの絶対的な値を知る必要があります。文字データが格納されている先頭アドレスが、この変数の値になるので、この変数の値STRPTRとVARPTRの相対アドレスの値を加えることによって、先頭アドレスを知ることができます。

サンプルプログラム

```
AD=STRPTR:PRINT AD;HEX$(AD)
-24889 9EC7
OK
```

3.6.3 DTL

機　能　現在読み込み中のDATA文の行番号を与えます。

書　式

```
DTL
```

引数はありません。

省略形　DT.

解　説　DATA文のデータを読み込んでいるとき、その行番号が、この変数の値になります。

サンプルプログラム

```
IF DTL=1000 THEN RESTORE 1500
```

3.6.4 ERL

機能 エラー発生時に、その行番号を与えます。

書式

```
ERL
```

引数はありません。

省略形 なし

解説 プログラムの実行中、エラーが発生した場合、そのエラーが発生した文の行番号が、この変数の値になります。なお、ダイレクト実行でのエラー発生では影響を受けません。

ON ERROR GOTO文のサンプルプログラム参照。

サンプルプログラム

```
LIST
10 ON ERROR GOTO 1000
20 PRINT 10/0
30 PRINT 1E+38*1E+38
40 END
1000 IF ERL=20 THEN PRINT"Division by zero":RESUME NEXT
1010 IF ERL=30 THEN PRINT"Over flow":RESUME NEXT
1020 ON ERROR GOTO 0
OK
RUN
Division by zero
Over flow
OK
```

3.6.5 ERR

機能 エラー発生時に、そのエラーコードを与えます。

書式

```
ERR
```

引数はありません。

省略形 なし

解説 プログラムの実行中、エラーが発生した場合、そのエラーのエラーコードが、この変数の値になります。エラーコードについては、「エラーコード表」を参照してください。なお、ダイレクト実行でのエラー発生では影響を受けません。

ON ERROR GOTO文のサンプルプログラム参照。

サンプルプログラム

```
LIST
10 A=10/0
OK
RUN
Division by zero in 10
OK
?ERR
 11
OK
?ERL
 10
OK
ERROR ERR
Division by zero
OK
```

付　　　録

A.1 テキスト画面とその属性ポートへのアクセス方法

テキスト画面とテキスト画面の属性は共に入出力ポートに割りつけられていて、それぞれ2Kバイト（2048バイト）の容量をもっており、そのアドレスは

テキスト画面　&H3000〜&H37FF

テキスト属性　&H2000〜&H27FF

となっています。

テキスト画面とテキストの属性とはちょうど1対1に対応していて、OUTコマンドやPOKE@ステートメントを使うことにより、1バイト単位でテキストの属性を指定することができます。

テキストの属性8ビットのビット構成は下に示す通りです。

7	6	5	4	3	2	1	0
HS	VS	CG	CF	CR	G	R	B

(1) B・R・G…カラー指定(⇒COLOR)

テキスト画面のカラーはビットB(青指定)、R(赤指定)、G(緑指定)のビットパターンの組み合わせにより、下の表のように決定されます。

ビット	2 G	1 R	0 B	
	0	0	0	黒
	0	0	1	青
	0	1	0	赤
	0	1	1	マゼンタ
	1	0	0	緑
	1	0	1	シアン
	1	1	0	黄
	1	1	1	白

(2) CR…キャラクタの反転指定(⇒CREV)

0のときテキストのキャラクタは標準モード(normal mode)

1のときテキストのキャラクタが反転モード(reverse mode)となります。

(3) CF…キャラクタの点滅指定(⇒CFLASH)

0のときテキストのキャラクタは標準モード(normal mode)

1のときテキストのキャラクタは点滅モード(flashing mode)となります。

(4) CG…ROM/RAMの切り換え指定(⇒CGEN)

本機はユーザ定義のキャラクタジェネレータRAM(RAMCG)とあらかじめテキストのキャラクタが書き込まれているROM(ROMCG)を持っています。

ROMCG，RAMCGをアクセスするコードはそれぞれ&H00〜&HFFまでの256が割り当てられています。(⇒キャラクタコード表)

テキストのキャラクタ表示コードがROMCGをアクセスするのか、RAMCGをアクセスするのかを決定するのがビットCGです。

0のときROMCGを

1のときRAMCGをアクセスします。

(6) VS・HS…キャラクタのサイズ指定（⇒CSIZE）

ビットVS, HSを下の表のようにセットすることにより、キャラクタのサイズを変えることができます。

ただしVSをセットする場合は、一行分セットして下さい。

ビット 7 HS	6 VS	
0	0	標準サイズ
0	1	垂直2倍
1	0	水平2倍
1	1	垂直水平2倍

数値データの内部表現と外部表現

SHARP HuBASICで扱う数値は、内部表現では2進浮動小数点形式で記述され、演算や比較もこの形式のまま実行されます。2進数表現の場合、数値は必ずしも正確ではないため、実数演算、比較などの処理では、誤差の発生、あるいは、外部表現(PRINT文などによって表示される表現)との食い違いの発生に注意する必要があります。

たとえば、ある演算の結果の値が、数学的には整数となる場合でも、演算の途中で整数以外の値を扱っている場合には、内部表現では必ずしも整数が得られるとは限りません。

数値データの内部表現の外部表現に与える影響について例をあげて説明してみましょう。たとえば、変数Aの初期値を0として0.1ずつ加え続けるという演算を行うと、0.1は内部表現では正確な小数の0.1でないため、加え続けているうちに誤差を蓄積して、外部表現に影響を与えます。

プログラム1

```
10 A=0
15 FOR I=0 TO 1500
20 A=A+.1
30 PRINT A;                                注）.1=0.1
40 NEXT I
```

上のプログラムを実行すると、

```
RUN
 .1  .2  .3  .4  .5  .6  .7  .8  .9  1  1.1  1.2  1.3  1.4  1.5
1.6  1.7  1.8  1.9  2  2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8
2.9  3  3.1  3.2  3.3  3.4  3.5
                          ⋮
 44.7  44.8  44.9  45  45.1  45.2  45.3  45.4  45.5  45.6  45.7
45.8  45.899999  45.999999  46.099999  46.199999  46.299999
46.399999  46.499999  46.599999  46.699999  46.799999  46.899999
46.999999  47.099999  47.199999  47.299999  47.3
                          ⋮
```

というように初めは.1，.2，.3，…と増えて行きますが、45.8を過ぎると、誤差が現れます。

これは、内部表現の誤差がたまって、外部表現に影響を与えたために起こる現象です。

この不合理を取り除くための対策としては、まず第1に整数の1を加えるという整数演算を行って、その結果を10で割って小数に戻すという方法があります。

プログラム2

```
10 A=0
15 FOR I=0 TO 1500
20 A=A+1
30 PRINT A/10;
40 NEXT I
```

どうしても0.1ずつ加えたい場合には、10倍して整数型に変換してから10で割った値を出力するという方法や、いったん文字型に変換してから数値型に戻した値を出力するという方法があります。下にプログラムを示しますので各自試してみてください。

プログラム3（10倍した値をCINTを使って整数型に変換してから10で割る）

```
LIST
10 A=0
15 FOR I=0 TO 1500
20 A=A+.1
25 A=CINT(A*10)/10
30 PRINT A;
40 NEXT
OK
```

プログラム4（STR$を使って文字型に変換してからVALを使って数値型に戻す）

```
LIST
10 A=0
15 FOR I=0 TO 1500
20 A=A+.1
25 A=VAL(STR$(A))
30 PRINT A;
40 NEXT
OK
```

また、FOR～NEXT文において、プログラム5のように、初期値0から終了値1まで増加0.1で回すと、誤差のため、最後の1になるまでループしませんので、これを防ぐためにプログラム6のように、増分を整数の1として、初期値0から終了値10まで回すようにする必要があります。

プログラム5

```
LIST
10 FOR I=0 TO 1 STEP .1
20  PRINT I;
30 NEXT
OK
RUN
 0  .1  .2  .3  .4  .5  .6  .7  .8  .9
OK
```

プログラム6

```
LIST
10 FOR I=0 TO 10
20  PRINT I/10;
30 NEXT
OK
RUN
 0  .1  .2  .3  .4  .5  .6  .7  .8  .9  1
OK
```

そのほか、0.1を3で割って、次に3倍するという演算を行うと、内部表現が正確な0.1とならないためIF文の比較で演算結果と0.1が一致しないということが起こります。

プログラム7

```
LIST
10 A=.1/3*3
20 B=.1
30 PRINT"A=";A:PRINT"B=";B
40 PRINT USING"###.############";A;B
50 IF A=B THEN PRINT"ﾖｲ!":END ELSE PRINT"ﾜﾙｲ!":END
OK

RUN
A= .1
B= .1
  0.100000000035  0.100000000006
ﾜﾙｲ!
OK
```

IF文の比較は内部表現によっているため、このような不合理が生じるもので、これを解消するために、プログラム8のように比較の部分を変えて、10^{-8}の精度で比較するようにするとよいでしょう。

プログラム8

```
LIST
10 A=.1/3*3
20 B=.1
30 PRINT"A=";A:PRINT"B=";B
40 PRINT USING"###.############";A;B
50 IF ABS(A-B)<.00000001 THEN PRINT"ﾖｲ!":END ELSE PRINT"ﾜﾙｲ!":END
OK

RUN
A= .1
B= .1
  0.100000000035  0.100000000006
ﾖｲ!
OK
```

A.2 コマンド・ステートメント・関数の省略形一覧表

実際プログラミングをするとき、省略形を知っていると、たいへん便利なので、下にその一覧表を示します。

（注意） たとえば、LISTの省略形はL.ですが、実際にはそれより長いLI.やLIS.も省略形として使うことができます。そこで、下に示す省略形はいずれも省略形の中で最短のものとします。また、省略形の最後には必ずピリオド.をつけます。

省略形入力後、再び表示すると、もとの形で表示されます。

もとの形	省略形	もとの形	省略形
ABS	AB.	CURSOR	CU.
APSS	AP.	CVD	なし
ASC	なし	CVI	なし
ASK	AS.	CVS	CV.
ATN	AT.	DATA	DA.
AUTO	A.	DATE$	DATE.
BEEP	BE.	DAY$	DAY.
BIN$	BI.	DEFCHR$	DEFCHR.
BOOT	BO.	DEFDBL	DEFD.
CALL	CA.	DEFFN	なし
CANVAS	CAN.	DEFINT	DEFI.
CDBL	CD.	DEFKEY	DEFK.
CFLASH	CF.	DEFSNG	DEFS.
CGEN	CG.	DEFSTR	DEFST.
CGPAT$	CGP.	DEFUSR	なし
CHNNEL	CHAN.	DELETE	D.
CHARACTER$	CHAR.	DEVICE	DEV.
CHR$	CHR.	DEVI$	なし
CINT	CIN.	DEVO$	DEVO.
CIRCLE	CI.	DIM	DI.
CLEAR	CLE.	DTL	DT.
CLICKON	CLI.O.	EDIT	E.
CLICKOFF	CLI.OF.	EJECT	EJ.
CLOSE	CLO.	END	EN.
CLR	なし	EOF	EO.
CLS	CL.	ERASE	ER.
CMT	CM.	ERL	なし
COLOR	COL.	ERR	なし
CONSOLE	CONS.	ERROR	ERR.
CONT	C.	EXP	EX.
COS	なし	FAC	なし
CREV	CR.	FAST	FA.
CRT	なし	FILES	FIL.
CSIZE	CS.	FIX	なし
CSNG	CSN.	FN	なし
CSRLIN	CSR.	FOR	F.
CSTOP	CST.	FRAC	FR.

もとの形	省略形
FRE	なし
GET@	なし
GOSUB	GOS.
GOTO	G.
GRAPH	GR.
HCOPY	H.
HEXCHR$	HEXC.
HEX$	HE.
IF/THEN/ELSE	IF/TH./EL.
INKEY$	INK.
INP	なし
INPUT	I.
INPUT#	I.#
INPUT$	I.$
INSTR	INS.
INT	なし
KANJI$	KA.
KEY	K.
KEY0	K.0
KEYLIST	K.L.
KEYOFF	K.
KEYON	K.O.
KEYSTOP	K.S.
KILL	KI.
KLIST	KL.
LABEL	LA.
LAYER	LAY.
LEFT$	LEF.
LEN	なし
LET	(完全省略)
LFILES	LF.
LIMIT	LIM.
LINE	なし
LINE INPUT	LINEI.
LINE INPUT#	LINEI.#
LINPUT	LI.
LIST	L.
LLIST	LL.
LOAD	LO.
LOADM	LO.M
LOAD?	なし
LOG	なし
LOCATE	LOC.
LPOS	LPO.
LPRINT	LP.またはL?
MAXFILES	MA.
MEM$	MEM.

もとの形	省略形
MERGE	M.
MID$	MI.
MIRROR$	MIR.
MKD$	MKD.
MKI$	MK.
MKS$	MKS.
MON	MO.
MUSIC	MU.
NEW	なし
NEWON	なし
NEXT	N.
OCT$	OC.
ON	O.
OPEN	OPE.
OPTION BASE	OP.B.
OPTION SCREEN	OP.SC.
OUT	OU.
PAI	なし
PAINT	PAI.
PALET	PAL.
PATTERN	PAT.
PAUSE	PA.
PEEK	PE.
PEEK@	PE.@
PLAY	PL.
POINT	POI.
POKE	PO.
POKE@	PO.@
POLY	POL.
POS	なし
POSITION	POS.
PRESET	PRE.
PRINT	P. または?
PRINT#	P.#または?#
PRW	なし
PSET	PS.
PUT@	なし
RAD	なし
READ	REA.
REM	'(アポストロフィ)
RENUM	REN.
REPEAT	REP.
REPEAT OFF	REP.OF.
REPEAT ON	REP.O.
RESTORE	RES.
RESUME	RESU.
RETURN	RE.

もとの形	省略形
REW	なし
RIGHT$	RI.
RND	RN.
RUN	R.
SAVE	SA.
SAVEM	SA.M
SCREEN	SC.
SCRN$	SCRN.
SCROLL	SCRO.
SEARCH	SE.
SGN	SG.
SIN	SI.
SOUND	SO.
SPACE$	SPA.
SPC	なし
SPEED	SP.
SQR	SQ.
STICK	STI.
STOP	S.
STRIG	STR.
STR$	なし
STRING$	STRIN.
STRPTR	STRP.
SUM	なし
SWAP	SW.

〈論理演算子の省略〉
NOT=NO.
AND=AN.
XOR=X.
IMP =IM.
EQV=EQ.

もとの形	省略形
TAB	なし
TAN	TA.
TEMPO	TE.
TIME	TIM.
TIME$	なし
TRACE	T.
TROFF	TROF.
TRON	TRO.
TVPW OFF	TV.OF.
TVPW ON	TV.O.
UNTIL	U.
USR	なし
VAL	VA.
VARPTR	VAR.
VDIM	V.
VERIFY	VE.
VOL	VO.
WAIT	WA.
WEND	WE.
WHILE	W.
WIDTH	WI.
WINDOW	WIN.
WRITE	WR.
WRITE#	WR.#

A.3　機械語とのリンク方法

A.3.1　モニターのコマンド

本機のBASICは機械語の入力を容易にする為にMON部を設けています。

スタック・ワークエリアとしてはFF00～FFFF（16進）を使用しています。

このモニターはBASICのMON命令によって使用可能な状態になり、BASIC同様、スクリーンエディターを内蔵していて、次に示すエディット書式により、メインメモリー64Kのどのアドレスでも書きかえ可能です。

エディット書式

：アドレス＝データ_データ_データ……　　注）データ間の_は、1文字分のスペース(空白)を表わしています。

：（コロン）…エディット可能な行である事を示すマーク
（BASICの文番号とMONのアドレスを区別する為）

アドレス……16進数4文字以内で指定し、先頭のデータがはいるメインメモリーのアドレス（0～FFFF）

＝（等号）…アドレス部とデータ部を区別する為のセパレーター

データ………16進数2文字以内あるいは；（セミコロン）＋文字で指定し、8ビットのデータあるいは指定文字のキャラクターコードを指定のメモリーアドレスに書き込みます。データ間は原則として、スペースで区切りますが、データが2文字の場合は必要ありません。

*MFE00とキーインして例の様に実行してみて下さい。Aの文字を表示します。

(例)

：FE00＝3E_;A_CD_13_00_C9 ⏎
＊ SHIFT ＋ BREAK
＊GFE00

*P　プリンタースイッチ

D及びFコマンドの出力をプリンターにしたり画面にしたりします。

モニターにはいった時は画面モードとなっておりこのコマンドを実行するたびにモードが反転します。

プリンターがつながっていない場合はしばらくしてから、ERR？と出てコマンド待ちになりますから、プリンターをチェックするか。Pコマンドを実行して、画面モードにもどして下さい。

*D　ダンプ

*D[先頭アドレス[_最終アドレス]]

メモリーの内容を表示します。最終アドレスを省略すると先頭アドレスより、128バイトを表示します。先頭アドレスも省略すると、次のアドレスから128バイトを表示します。

次のフォーマットでダンプされます。

：HHHH＝HH_HH_HH_HH_HH_HH_HH_HH_/ABCDE.G.

（エデット可能マーク）（行先頭アドレス）（セパレーター）（8バイト16進データ）（セパレーター）(8バイト文字)

注）最後に出力される8バイト文字は、データのディスプレイコードでコントロールコードは.(ピリオド）で表示されます。

BREAK キーのみで表示ストップ SHIFT ＋ BREAK でコマンド待ちへもどります。

＊M　メモリーセット

＊M〔先頭アドレス〕

メモリーの内容をかきかえます。先頭アドレスを省略すると、現在のポインターからかきかえます。このモードからぬけるには、SHIFT ＋ BREAK を押して下さい。

アドレスとデータが表示されてカーソルはデータの上にきますので､エディットの書式でデータを指定して CR キーを押して下さい。指定のデータ数、アドレスが加算されて次の行へすすみます。

注） CR キーは ⏎ キーのことです。

＊F　ファインド

＊F　先頭アドレス␣最終アドレス␣データ␣データ␣…

先頭アドレスより最終アドレスまで、データで指定された個数の連続したデータをさがし、見つかったらそのアドレスとデータをダンプの様式で出力します。SHIFT ＋ BREAK で止まります。

＊G　ゴーサブ

＊G　コールアドレス

指定したコールアドレスをサブルーチンコールします。スタックポインターは、FFFE（16進）にあります。

＊T　トランスファ

＊T　先頭アドレス␣最終アドレス␣転送先頭アドレス

指定したアドレス間のデータを転送先頭アドレスからに転送します。

＊S　セーブ

＊S　先頭アドレス␣最終アドレス␣実行先頭アドレス：ファイル名

指定したアドレス間のデータをカセットテープに記録します。

実行先頭アドレスとはLOADした時の実行先頭アドレスです。

ファイル名は、BASICと同様次の様式で ：（コロン）の後に指定して下さい。年月日時分は自動的にタイマーより読み込まれ記録されます。

ファイル名の様式

：NAME. EXT

＊L　ロード

＊L　[ロード先頭アドレス][:ファイルネーム]

指定のファイルをカセットよりLOADします。先頭アドレスを指定すると、そのアドレスよりLOADし、指定しないと、SAVEした情報の通りLOADします。ファイルネームは指定しないと、最初に見つけたファイルをLOADします。途中でBREAKしたりチェックサムエラーが出た時は､ERR？と出てコマンド待ちにもどり､エラーが出なかった時はそのまま、コマンド待ちにもどります。

＊V　ベリファイ

＊V〔：ファイルネーム〕

指定のファイルをカセットより読み、メインメモリーの内容と比較します。正しくSAVEされたかどうかを調べる時に使い正しくない場合はERR？と出て止まります。

＊R　リターン

モニターをコールしたシステムへリターンします。

BASICのMON命令からはいった場合は、このコマンドによりBASICへもどれます。SP及びHLは保存していますのでMONの次の命令を実行します。

スタックポインター（SP）がFFOO～FFFFにあるようなシステムからコールされた場合や、スタックにリターンアドレスがはいってない場合は、この命令ではリターンできません。

Gコマンドより、そのシステムのホットスタートをCALLして下さい。

A.3.2 USR命令の使い方

BASICより機械語サブルーチンを呼び出す命令には、CALL命令とUSR命令があります。

この中でUSR命令はデータの受け渡しができる他、機械語サブルーチンからのエラー処理ができる等の機能をもっていますので次にその機能を説明します。

まずUSR命令を使う為には、そのアドレスを定義してやる必要があります。これにはDEFUSRを使います。

DEFUSR〔n〕=コールアドレス

nは0～9の番号でいくつものUSR命令を使用する時、最大10個まで定義できます。省略すると0です。

コールアドレスはユーザーの機械語サブルーチンがはいっているアドレスであり、メインメモリー64Kのどこにでも指定できます。

指定せずに使用する事はしないで下さい。定義を解除する命令はありませんので、前の定義がのこったままとなり、暴走する事があります。

DEFUSRで定義したら、次のようにして使います。

USR[n]({数　式 / 文字式})

例　A=USR(B)
A$=USR1(C$)
D=USR(65)

数式あるいは文字式で指定されたデータをもってユーザー定義n番のサブルーチンを呼び出し、リターンした時のデータをもつ関数となります。サブルーチン中でデータを書きかえるとその書きかえたデータをもつ関数となります。ただし、データのタイプは見る事はできても変える事はできません。

サブルーチンへ受けわたすデータ

IXレジスター…エラー処理のアドレスを示しています。
Accにエラー番号をセットしてJP(IX) を実行する事によりエラー処理ができます。

Accレジスター…データのタイプを示しています。
2……整数形
5……単精度
8……倍精度
3……文字形

HLレジスター… データのはいっているメインメモリー上のアドレスを示しています。文字形のみデータではなく、ストリングディスクリプターと呼ばれるエリアを示していますので、このデータでは、文字のデータはわかりません。DEレジスタとBレジスタで見て下さい。
データはHLのアドレスから次のようにはいっています。

整数形……………

Low	High

単精度……………

指数	MSB　仮数4バイト　LSB

倍精度……………

指数	MSB　仮数7バイト　LSB

文字形……………

長さ	オフセット値

※指数……………仮数部のMSBが指数を示しています。
81（16進）が2で、00の場合がデータ0を示します。

※仮数……………MSBを1とする2進データで必ずMSBが1である事より、このビットを正・負の符号ビットとして使い0の場合　正の数、1の場合負の数となります。

※オフセット値…ストリングエリアのスタートアドレスからのオフセットであり、サブルーチン中からこのアドレスを見ても意味はないので、DEレジスタを見て下さい。

DEレジスタ……Aレジスタが3つまり文字形の時のみ意味をもち、文字の先頭がDEレジスタ文字の長さがBレジ
及びBレジスタ　スタになっています。Bレジスタが0の時はDEレジスタは意味をもちません。データを書きなおす場合は、Bレジスタの長さ以上入れないで下さい。またBASICのUSR関数(　)の中が文字変数のみの場合は、直接変数のアドレスをもっていますので(　)の中の変数の値がかわってしまいます。文字式を渡す場合は、USR(A$+"")というように文字式の加算の形で渡して下さい。

A.4 コード表

A.4.1 キャラクタコード表（ASCIIコード表）

上位4ビット→

下位4ビット↓

/	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	`	p				ー	タ	ミ		
1			!	1	A	Q	a	q			。	ア	チ	ム		土
2			"	2	B	R	b	r			「	イ	ツ	メ	♠	金
3			#	3	C	S	c	s			」	ウ	テ	モ	♥	木
4			$	4	D	T	d	t			、	エ	ト	ヤ	♦	水
5			%	5	E	U	e	u			・	オ	ナ	ユ	♣	火
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		月
7			'	7	G	W	g	w			ァ	キ	ヌ	ラ		日
8			(	8	H	X	h	x			ィ	ク	ネ	リ		時
9			)	9	I	Y	i	y			ゥ	ケ	ノ	ル		分
A			*	:	J	Z	j	z			ェ	コ	ハ	レ		秒
B			+	;	K	[	k	{			ォ	サ	ヒ	ロ		年
C		→	,	<	L	¥	l	¦			ャ	シ	フ	ワ		円
D		←	-	=	M	]	m	}			ュ	ス	ヘ	ン		人
E		↑	.	>	N	^	n	‾			ョ	セ	ホ	゛		生
F		↓	/	?	O	_	o	π			ッ	ソ	マ	゜		〒

本機で使用するキャラクタ（文字）のコード表を上に示します。
キャラクタコードは16進数の００からＦＦ（&H00～&HFF）まであります。
上の表は横の数字が16進数の上位桁、縦の数字が下位桁を表しています。
たとえば、キャラクタＳのコードは&H53となります。

&H00～&H1Fは、コントロールキャラクタで、プログラム中でこれらを使うにはCHR$関数を用います。
たとえば&H1C（[CTRL]＋[¥]）はカーソルを右に移動し、&H07（[CTRL]＋[G]）はポッとブザーを鳴らします。
単なる文字として表示させるには、PRINT＃０文を利用してください。
&HA0はスペースと同じように１文字の空白ですが、スペースとは区別されます。
それを打ち込むにはカナモードにしてから、[SHIFT]キーを押しながら[ロ]キー（カタカナのロ）を打ってください。

A.5エラーメッセージ一覧表

エラーコード	エラーメッセージ	説　　明
1	NEXT without FOR	FORがないのにNEXTがある。
2	Syntax error	文法がまちがっている。
3	RETURN without GOSUB	GOSUBがないのにRETURNがある。
4	Out of DATA	READで読むべきデータがDATA文に用意されていない。
5	Illegal function call	規定外の数値やデータが指定されている。
6	Overflow	演算結果が許容範囲を越えた。
7	Out of memory	プログラムが大きすぎる。配列などの変数を多くとりすぎている。
8	Undefined label	GOTO, GOSUB, IFなどで指定した分岐先の行番号がない。
9	Subscript out of range	配列変数の添字が規定外である。
10	Duplicate Definition	配列が2重に定義されている。
11	Division by zero	0で割った。
12	Illegal direct	ダイレクト実行できないステートメントを実行しようとした。
13	Type mismatch	変数の型が一致しない。
14	―――	―――
15	String too long	文字が255文字を越えている。
16	Too complex	式が複雑すぎる。たとえば、(　)が異常に多い場合。
17	Can't continue	CONTによってプログラムの実行を再開できない。
18	Undefined function	DEFで定義されていない関数を呼んだ。
19	No RESUME	RESUMEによってプログラムの実行を再開できない。
20	RESUME without error	エラーがないのにRESUMEを実行しようとした。
21	Illegal format	エラーメッセージの定義されていないエラーが起こった。
22	Missing operand	パラメータの必要な命令に指定がない。
23	Line buffer overflow	1行の入力文字が多すぎる。
24	―――	―――
25	Bad screen mode	グラフィックメモリーを外部記憶として使おうとした。(WIDTH80でASKを実行した。)
26	UNTIL without REPEAT	REPEATがないのにUNTILを実行しようとした。
27	Out of tape	カセットテープがセットされていない。
28	―――	―――
29	Tape read ERROR	カセットテープからデータが正しく読めない。
30	Bad file mode	異ったモードのファイルを参照しようとした。(ファイルにはバイナリィ、Basicテキスト、アスキータイプの3つのモードがある。)
31	Out of stack	POPを実行しようとしたがスタックに何も入っていない。
32	WHILE without WEND	WHILEループにWENDがない。
33	WEND without WHILE	WHILEがないのにWENDがある。
34	Reserved feature	ディスクBASIC用のコマンドを実行しようとした。
35	FOR without NEXT	FORループにNEXTがない。
36	Format over	PRINT USINGで指定したフォーマットが長すぎて出力できない。
37	REPEAT without UNTIL	REPEATループにUNTILがない。
50	FIELD overflow	FIELD文でランダムファイル内のレコード長が256以上になっている。
51	Device in use	外部装置の使用中である。
52	Bab file number	オープンされていないファイルや、起動時に指定しないファイルを参照しようとした。
53	File not found	LOAD,KILL,OPENでディスクにないファイルを参照しようとした。
54	Already open	すでにOPENしているファイルを再びOPENしようとした。
55	―――	―――

エラーコード	エラーメッセージ	説　明
56	Device I/O ERROR	入出力装置において入出力エラーが生じた致命的エラー。
57	File already exists	NAMEで変更しようとしたファイル名がすでに登録されている。
58	――	――
59	――	――
60	Device full	データが入出力装置の許容容量を越えた。
61	Input past end	end of file のファイルを読もうとしたか、空ファイルを読もうとした。
62	――	――
63	――	――
64	Bad allocation table	ディスケット中のFATが壊れている。
65	Bad file descripter	ディスクリプタが違う。
66	Bad record	レコード番号が規定外。
67	No password	パスワードがあわない。
68	――	――
69	――	――
70		
71	File not open	OPENされていないファイルを使用しようとした。
72	Write protected	指定されたカセットテープの消去防止用のツメがとれている。
73	Device offline	入出力装置がつながっていないのに使用しようとした。

注）ここに登録されていない上記以外のエラーメッセージは、Unprintable error と表示されます。

A.6 ファイルディスクリプタ表

		使用できるモード
SCR:	画面(コントロールコードを実行する)	O
CRT:	画面(コントロールコードを表示する)	O
KEY:	キーボード	I
LPT:	プリンター	O
CAS:	カセットファイル	IまたはO
MEM:	グラフィックメモリー	IまたはO
EMM0: ～ EMM9:	外部メモリー	IまたはO

SHARP HuBASICで使用するファイルディスクリプタ表を上に示します。
ファイルディスクリプタは、外部デバイスを指定する文字定数で、最後に：(コロン)をつけるのを忘れないでください。

A.7 メモリーマップ

アドレス	メモリー領域	注記
0000	IOCS	←&H0000
	モニター	
	BASIC インタープリタ	
	テキスト	←NEW ONで指定したテキストアドレス
		→テキストエンドコード"0 0　0 0"
	変数エリア	←VARPTR
		→変数エンドコード"0 0"
	ファイル用ストリングバッファ ＃0, ＃1, …	←STRPTR / MAXFILESnでとられるエリア (n＋1)×(16+256) バイト
	ストリングデータバッファ	
	テンポラリーストリングエリア	
	↓ フリーエリア ↑	
	STACK	
	FAC	←(CLEAR, LIMIT)－&H100
	ユーザー用機械語フリーエリア	←CLEAR, LIMITで指定したアドレス
FF00	IPL用ワークエリア	←&HFF00
FFFF		←&HFFFF

A.8 SHARP HuBASICプログラマのためのワンポイントアドバイス

1. INPUT文などでデータを入力した後や、プログラムをエディットした後には必ず [⏎] キーを押すようにしましょう。
2. LIST, RUN, LOAD, SAVEなどを実行する前には必ず [SHIFT] [CLR HOME] を押して画面をクリアしてから行うように習慣づけてください。
3. カーソルを画面の下方向に動かすときには必ず [⇩] キーを使うようにし、[⏎] キーはデータやプログラムを入力するときにだけ使うようにしましょう。
4. 画面が複雑になったり、カーソルが見えなくなったりしてどうしようもなくなったときには、次の順番で操作を行ってください。
   (1) ディスプレイテレビの電源は大丈夫か確認する。
   (2) ディスプレイテレビとコンピュータとの接続部分は大丈夫か確認する。
   (3) ディスプレイテレビがコンピュータモードになっているか確認する。
   (4) キーボード上の [SHIFT] [BREAK] キーを押す。
   (5) [CTRL] [D] を押す。
   (6) [SHIFT] [CLR HOME] を押す。
   (7) [S] [C] [・] [⏎] を押す。
   (8) 本機の背面のリセットスイッチを押す。
   (9) 本機の電源をいったん切って、5秒後に再び入れる。
5. プログラムを能率的に作成するには、次のキー操作に精通するとよいでしょう。
   (1) [CTRL] [A] …INS（インサート）モードの設定、解除を行う。
   (INSモード…文字列を間に挿入する。)
   (2) [CTRL] [E] …カーソルから右を行の終わりまで消す。
   (3) [CTRL] [Z] …カーソル以降のテキスト画面をすべてクリアする。
   (4) [⇧]、[⇩]、[⇦]、[⇨] を使ってカーソルの移動を行う。
   (5) [SHIFT] [CLR HOME] …画面を全てクリアする。
6. CLEARはサブルーチン内で実行できないので注意してください。メインプログラムの最初に実行するのが常識です。
7. プログラムで小数を使わない場合は、最初にDEFINT A－Zで変数の整数型宣言をしておくと、プログラムの実行速度が上がります。
8. FOR～NEXT文でNEXTの後の変数を省略するとループ速度が速くなります。
9. TRONは画面が乱れるだけでデバッグ効果はあまり期待できません。むしろ、プログラムの随所にSTOPを入れてデバッグするほうがよいでしょう。
10. プログラムの先頭にINIT文を入れるようにしましょう。
11. ON KEY GOSUB文などでファンクションキーが効かなくなったときは [O] [⏎] を押せば動くようになります。
12. TIME$は1度設定したらさわらないで、時間の計測にはTIMEを使いましょう。
13. プログラムの実行中に [BREAK] キーのみ押すと一時停止します。はなすと実行を再開します。
14. APSSの動作を確実にするため、カセットテープは15分テープを使って下さい。
    また、すでに記録してあるテープを再使用する場合には、必ず消去してから使用して下さい。
15. プログラム実行中にエラーが発生したときには、[E] [・] [・] [⏎] と押すと、便利にエディットできます。
16. カセットが作動している間、キーからの入力を受けつけにくくなる場合がありますが、カセットの動作が終わると、一気に入力されます。
17. プログラムはなるべく1本のカセットテープに1つの割合でSAVEすると、プログラムをLOADするとき速くて便利です。
18. プログラムは同じものを2回続けてカセットテープにSAVEするように心がけてください。1つが壊れても、もう1つが使えます。カーソルをSAVE文のところにもっていって [⏎] キーを押せば簡単です。
19. カセットデッキのフタを閉じるときは静かにフタの左側を押すようにしてください。
20. ファイル名として ,（カンマ）:（コロン）;（セミコロン）は使用できません。

A.9 サンプルプログラム集

A.9.1 キャラクタ定義 1

```
10 REM SAMPLE PROGRAM
20 SCREEN0,0,0:OPTION SCREEN1:CGEN:WIDTH40
:COLOR7,0:WINDOW:PALET:CSIZE:CREV:CFLASH:PRW
30 FOR I=0 TO 255
40 DEFCHR$(I)=STRING$(3,LEFT$(CGPAT$(I),8))
50 NEXT
60 CGEN 1:CSIZE 0
70 FOR I=0 TO 255
80 PRINT #0,CHR$(I);
90 NEXT
100 FOR I=0 TO 255
110 DEFCHR$(I)=LEFT$(STRING$(3,LEFT$(CGPAT$(I),8)),23)+CHR$(85)
120 NEXT
130 CSIZE
```

A.9.2 キャラクタ定義 2

```
10 REM SAMPLE PROGRAM
20 SCREEN0,0,0:WIDTH 40:CREV:CFLASH:COLOR 7,0:CLS4:CGEN:CSIZE
30 CGEN 1:CSIZE 0
40 FOR I=0 TO 255
50 PRINT #0,CHR$(I);
60 NEXT I
70 FOR I=0 TO 255
80 DEFCHR$(I)=LEFT$(STRING$(3,LEFT$(CGPAT$(I),8)),23)+CHR$(255)
90 NEXT I
```

ユーザー定義文字に、ノーマル文字に、下線を引いたパターンを定義します。

A.9.3 チェック模様と斜線

```
10 REM SAMPLE PROGRAM
20 SCREEN0,0,0:WIDTH 80:COLOR7,0:WINDOW:
PALET:CSIZE:CREV:CFLASH:PRW:CLS4
30 COLOR2:A$=HEXCHR$("55AA")
40 FOR I=0 TO 99
50 POSITION0,I*2
60 FOR J=0 TO 79
70 PATTERN-2,A$
80 NEXT J
90 NEXT I
100 COLOR5:A$=HEXCHR$("0100040010004000")
110 FOR I=0 TO 24
120 POSITION0,I*8
130 FOR J=0 TO 79
140 PATTERN-8,A$
150 NEXT J
160 NEXT I
170 END
```

一回目はチェック模様が表示され、二回目は斜線が表示されます。

A.9.4 線と図

```
10 REM SAMPLE PROGRAM
20 SCREEN0,0,0:WIDTH40:COLOR7,0:WINDOW:
PALET:CSIZE:CREV:CFLASH:PRW:CLS4
30 X0=INT(RND(1)*320)
40 Y0=INT(RND(1)*200)
50 X1=INT(RND(1)*320)
60 Y1=INT(RND(1)*200)
70 C=INT(RND(1)*7)+1
80 LINE(X0,Y0)-(X1,Y1),PSET,C,BF
90 X0=X1:Y0=Y1
100 GOTO 50
```

画面につながった線を引きます。
また、80行をとりかえると中間色で四角形を書きます。

```
80 LINE(X0,Y0)-(X1,Y1),PSET,BF,HEXCHR$("AAFFEE55FFBB")
```

A.9.5 ペイントデモプログラム

```
10 REM SAMPLE PROGRAM
20 SCREEN 0,0,0:WIDTH80:COLOR7,0:WINDOW:
PALET:CSIZE:CREV:CFLASH:PRW:CLS4
30 FORI=0TO7:CIRCLE(I*80+40,50),38,I,1,0,360:
LINE(I*80,100)-(I*80+79,199),PSET,7,B:
PAINT(I*80+40,50),I,I:NEXT
40 PAINT(81,101),HEXCHR$("AA0000550000"),7:
PAINT(161,101),HEXCHR$("00AA00005500"),7:
PAINT(241,101),HEXCHR$("AAAA00555500"),7
50 PAINT(321,101),HEXCHR$("0000AA000055"),7:
PAINT(401,101),HEXCHR$("AA00AA550055"),7:
PAINT(481,101),HEXCHR$("00AAAA005555"),7:
PAINT(561,101),HEXCHR$("AAAAAA555555"),7
60 END
```

円を色つきで表示し、その内部をぬりつぶし、長方形を書いてその内部をすこしうすい色でぬりつぶします。

A.9.6 ペイントタイルパターンデモプログラム

```
10 REM SAMPLE PROGRAM
20 SCREEN 0,0,0:OPTION SCREEN1:CGEN:WIDTH80:COLOR7,0:
WINDOW:PALET:CSIZE:CREV:CFLASH:PRW
30 CLS4:LOCATE 2,10:PRINT"AUTO or MANUAL(A/M)?";
40 A$=INKEY$(1):IF A$="A"THEN FL=1 ELSE FL=0
50 FOR I=0 TO 7
60 LINE(I*160+40,20)-(I*160+119,59),PSET,7,B
70 NEXT
80 RESTORE 250
90 FOR I=0 TO 3
100 IF FL=1 THEN KEY0,STR$(I+1)+CHR$(13)
110 LOCATE I*20+2,11:INPUT"ﾀﾞﾝｽｳ",A
120 PAUSE 250:PAUSE 250
130 IF A<1 OR A>8 GOTO 110
140 A$=""
150 FOR J=1 TO A
160 IF FL=1 THEN READB$:KEY0,B$+CHR$(13)
170 LOCATE I*20+2,11+J:INPUT "ﾊﾟﾀｰﾝ",B$:B$=HEXCHR$(LEFT$(B$,6))
180 PAUSE 250:PAUSE 250
190 IF LEN(B$)<>3 GOTO 170
200 A$=A$+B$
210 NEXT
220 PAINT(I*160+41,21),A$,7
230 NEXT I
240 GOTO 30
250 DATA AAFFFF,AAFF55,55FFAA,222200,555500,888800,
010001,020002,040004,080008
```

ペイントのタイルパターンを見るプログラムです。AUTO or MANUALとたずねてくるので、大文字でAかMを押して下さい。Aを押すと自動で段数、パターンが入力されます。Mを押すと段数の入力です。1から8まで入力して下さい。次にパターンです。16進数6ケタの入力で、段数分入力がくりかえされます。

A.9.7 星

```
10 REM SAMPLE PROGRAM
20 SCREEN0,0,0:WIDTH 40:COLOR7,0:WINDOW:
PALET:CSIZE:CREV:CFLASH:PRW:CLS4
30 FOR X=26 TO 319 STEP52
40 FOR Y=20 TO 199 STEP40
50 C=INT(RND(1)*7)+1
60 POLY(X,Y),28,C,144,90,810
70 NEXT Y
80 NEXT X
```

画面に星型を色つきで表示します。

A.9.8 三原色

```
10 REM SAMPLE PROGRAM
20 SCREEN0,0,0:WIDTH 40:COLOR 7,0:WINDOW:PALET:CSIZE:
CREV:CFLASH:PRW:CLS4
30 CIRCLE(160,70),50,7,1,0,360:CIRCLE(134,115),50,7,1,0,
360:CIRCLE(186,115),50,7,1,0,360
40 PAINT(160,50),1,7:PAINT(117,125),2,7:PAINT(203,125),4,
7:PAINT(117,75),3,7:PAI
NT(160,150),6,7:PAINT(203,75),5,7:PAINT(160,100),7,7
50 PAINT(117,125),2,7:PAINT(203,125),4,7:PAINT(117,75),3,
7:PAINT(160,150),6,7:PAINT(203,75),5,7:PAINT(160,100),7,7
60 CIRCLE(160,70),50,1,1,0,180:CIRCLE(134,115),50,2,1,
120,300:CIRCLE(186,115),50 ,4,1,-120,60:CIRCLE(160,70),50,
3,1,181,240:CIRCLE(134,115),50,6,1,300,360:CIRCLE(186,115)
,50,5,1,60,120
70 CIRCLE(160,70),50,5,1,300,360:CIRCLE(134,115),50,3,1,
60,120:CIRCLE(186,115),50,6,1,180,240
80 LOCATE 15,11:PRINT"■■\//■■":LOCATE 15,12:PRINT"■
oo  ■":LOCATE15,13:PRINT"■■/\\■■":CANVAS1,2,4
90 LAYER1,2,3,4:PAUSE 10:LAYER1,2,4,3:PAUSE 10:LAYER1,3,2
,4:PAUSE 10:LAYER1,3,4,2:PAUSE10:LAYER1,4,2,3:PAUSE10:LAY
ER1,4,3,2:PAUSE10
100 LAYER2,1,3,4:PAUSE 10:LAYER2,1,4,3:PAUSE 10:LAYER2,3,
1,4:PAUSE 10:LAYER2,3,4,1:PAUSE10:LAYER2,4,1,3:PAUSE10:
LAYER2,4,3,1:PAUSE10
110 LAYER3,1,2,4:PAUSE 10:LAYER3,1,4,2:PAUSE 10:LAYER3,2,
1,4:PAUSE 10:LAYER3,2,4,1:PAUSE10:LAYER3,4,1,2:PAUSE10:L
AYER3,4,2,1:PAUSE10:GOTO 90
```

画面に色のちがう丸を表示し、その中央に字を入れます。そして、優先順位をいろいろ変えています。

A.9.9 カエルの歌

```
10 REM SAMPLE PROGRAM
20 REM FROG FROG FROG
30 SCREEN0,0,0:OPTION SCREEN1:CGEN:WIDTH40:COLOR7,
0:WINDOW:PALET:CSIZE:CREV:CFLASH:PRW
40 TEMPO 120
50 MUSIC"O4V10:O5V8:O3V12"
60 MUSIC"C5DEFEDCR:R5RRRRRRR:R5RRRRRRR"
70 MUSIC"CDEFEDCR:EFGAGFER:RRRRRRRR"
80 MUSIC"CDEFEDCR:EFGAGFER:CRCRCRCR"
90 MUSIC"CDEFEDCR:EFGAGFER:CRCRCRCR"
100 MUSIC"CDEFEDCR:RRRRRRRR:CRCRCRCR"
110 MUSIC"CDEFEDCR:EFGAGFER:RRRRRRRR"
120 MUSIC"CDEFEDCR:RRRRRRRR:RRRRRRRR"
130 MUSIC"R:R:R"
140 GOTO 60
```

A.9.10 TEL.サウンド

```
10 REM SAMPLE PROGRAM
20 REM TELEPHON
30 SCREEN0,0,0:OPTION SCREEN1:CGEN:WIDTH40:COLOR7,0:
WINDOW:PALET:CSIZE:CREV:CFLASH:PRW
40 TEMPO 300
50 MUSIC"E0A0EAEAEAEAEAEAEAEAEAEAEAEAEAEAEAEAEAEAEAE
AEAR9R9R9"
60 GOTO 50
```

A.9.11 D51 サウンド

```
10 REM SOUND
20 SOUND0,16:SOUND1,2:SOUND2,0:SOUND3,2:SOUND4,32:SOUND5,
2:SOUND6,31:SOUND7,28:SOUND8,31:SOUND9,31:SOUND10,31:SOU
ND11,6:SOUND12,5:SOUND13,10
30 I=1500:J=-2
40 H=I ¥ 256:L=I AND 255
50 IF L<>0 THEN SOUND 11,L
60 IF H<>0 THEN SOUND 12,H
70 I=I+J:IF I<500 OR I>1500 THEN J=-J
80 GOTO 40
```

A.9.12 サイコロ 1

```
10 REM SAMPLE PROGRAM
20 SCREEN0,0,0:WIDTH 40:COLOR 7,0:WINDOW:PALET:CSIZE:CREV:
CFLASH:PRW:CLS4
40 CLS 0
50 WINDOW (80,50)-(239,149),(0,0)-(319,199)
60 GOSUB 120
70 FOR I=0 TO 3:FOR J=0 TO 3
80 IF (I MOD 3)<>0 AND(J MOD 3)<>0 GOTO 100
90 WINDOW(I*80,J*50)-(I*80+79,J*50+49),(0,0)-(319,199):
GOSUB 120
100 NEXT J:NEXT I
110 END
120 POLY(160,100),90,7,60,30,390:POLY(160,100),90,7,0,30:
POLY(160,100),90,7,0,15 0:POLY(160,100),90,7,0,270
130 PAINT(160,55),HEXCHR$("FFAA55FF55AA"),7:PAINT(121,122),
HEXCHR$("AAFF5555FFAA "),7:PAINT(198,122),
HEXCHR$("55AAFFAA55FF "),7:RETURN
```

中央に、ウィンドウを使ってすこし小さくしたハコを書きます。
また、ウィンドウを使ってもっと小さいハコをそのまわりに書きます。

A.9.13 サイコロ　2

```
10 REM SAMPLE PROGRAM
20 SCREEN0,0,0:WIDTH 40:COLOR 7,0:WINDOW:PALET:CSIZE:
CREV:CFLASH:PRW:CLS4
40 CLS 0 :PALET 7,0
50 WINDOW (80,50)-(239,149),(0,0)-(319,199)
60 GOSUB 120
70 FOR I=0 TO 3:FOR J=0 TO 3
80 IF (I MOD 3)<>0 AND(J MOD 3)<>0 GOTO 100
90 WINDOW(I*80,J*50)-(I*80+79,J*50+49),(0,0)-(319,199):
GOSUB 120
100 NEXT J:NEXT I
110 FOR I%=0 TO 7:PALET I%,I%:PALET(I%-1)-(I%=0)*8,0:
PAUSE 50:NEXT :GOTO 110
120 POLY(160,100),90,7,60,30,390:POLY(160,100),90,7,0,30:
POLY(160,100),90,7,0,15 0:POLY(160,100),90,7,0,270
130 PAINT(160,55),HEXCHR$("FFAA55FF55AA"),7:PAINT(121,122)
,HEXCHR$("AAFF5555FFAA "),7:PAINT(198,122),
HEXCHR$("55AAFFAA55FF "),7:RETURN
```

画面にハコを書き、パレットを使い色をかえ、まるでストロボの光があたっているように見えます。

A.9.14 サイコロ　3

```
10 REM SAMPLE PROGRAM
20 SCREEN 0,0,0:WIDTH 40:COLOR 7,0:WINDOW:
PALET:CSIZE:CREV:CFLASH:PRW:CLS4
30 CLS 0:PALET 7,0
40 WINDOW (80,50)-(239,149),(0,0)-(319,199)
50 GOSUB 140
60 FOR I=0 TO 3:FOR J=0 TO 3
70 IF (I MOD 3)<>0 AND(J MOD 3)<>0 GOTO 90
80 WINDOW(I*80,J*50)-(I*80+79,J*50+49),(0,0)-(319,199):GOSUB 140
90 NEXT J:NEXT I
100 FOR I=0 TO 7:PALET I,0:NEXT
105 WINDOW
110 Y=1:Z=1:PRW193
120 LINE(0,Y+Z)-(39,Y+Z),"■",BF:PAUSE10:LINE(0,Y-Z)-(39,Y-Z)
," ",BF:Y=Y+Z:IF Y=1 OR Y=23 THEN Z=-Z
130 GOTO 120
140 POLY(160,100),90,7,60,30,390:POLY(160,100),90,7,0,30:
POLY(160,100),90,7,0,150:POLY(160,100),90,7,0,270
150 PAINT(160,55),HEXCHR$("FFAA55FF55AA"),7:PAINT(121,122),
HEXCHR$("AAFF5555FFAA"),7:PAINT(198,122),HEXCHR$
("55AAFFAA55FF"),7:RETURN
```

いくつかのハコを書いて、それを黒くします。そこに3行の白いキャラクターを書き、優先順位を指定して、ハコのあった所を黒くなるように見せています。

BASICテープのコピー作成方法

X 1に付属のカセットテープ「SHARP HuBASIC CZ8CB01」は、テープの長時間使用による摩耗、あるいは不慮の事故による破損によって使用できなくなることが考えられます。あらかじめ予備のBASICテープを作成しておくようにしてください。

BASICテープをコピーするには、コピーのためのプログラムを作る必要がありますので、その手順を次に示します。

①BASICテープをデッキにセットして、「SHARP HuBASIC CZ8CB01」を起動した後、テープを巻き戻したままにしておきます。

②下に示すBASICプログラムをキーボードから入力します。

```
10 '
20 '  -  BASIC ﾃｰﾌﾟ ｺﾋﾟｰ ﾌﾟﾛｸﾞﾗﾑ
30 '
40 CLEAR &HFE00
50 RESTORE 100
60 FOR I=0 TO 49
70 READ A$:POKE &HFE00+I,VAL("&H"+A$)
80 NEXT
90 CALL &HFE00
100 DATA 21,60,FE,01,20,00,CD,41,00,2A
110 DATA 74,FE,ED,4B,72,FE,CD,44,00,3E
120 DATA 01,CD,1B,00,FE,20,20,F7,21,60
130 DATA FE,01,20,00,CD,3B,00,2A,74,FE
140 DATA ED,4B,72,FE,CD,3E,00,C3,13,FE
```

③上のプログラムをRUNすると、自動的にCZ8CB01がロードされ、終了後カーソルが点滅し、入力待ちになります。

④カセットデッキに、巻き戻してある新しい空テープをセットしてから、キーボード上のスペースキーを押すと、CZ8CB01のセーブが始まります。

⑤セーブが終了すると、カーソルが点滅し始めますが、そのままの状態にしておくと、テープが空回りし続けますので、キーボード上のカセットストップキーを押して止めてください。

⑥セーブが終了した時点で、④の操作を繰り返すことにより、何本でもコピーを取ることができます。

⑦プログラムの実行を止めるときは、X 1の背面にある黒いRESETボタンを押してプログラムを消し、BASIC起動時の状態に戻してください。

⑧コピーテープを作成したら、消去防止用のツメを折って、大切に保管してください。